

SensMat Particle Sensor

Generated by Doxygen 1.9.1

1 SensMat Particle Sensor <!-- omit in toc -->	1
1.1 Description	2
1.1.1 Features	2
1.1.2 Documentation	2
1.1.3 Program Flow Chart	2
1.2 Usage	3
1.2.1 Interfaces	4
1.2.2 Starting the sensor	5
1.2.3 Charging the sensor	5
1.2.4 SD card logging	5
1.2.5 USB serial command interface	5
1.2.5.1 Supported commands	6
1.2.5.2 RTC Calibration and Settings	7
1.3 Development	7
1.3.1 Preparations	7
1.3.2 Toolchain	7
1.3.2.1 Option 1: mbed-cli (tested on Linux)	8
1.3.2.2 Option 2: SW4STM32 (Eclipse with build tools, tested on Windows)	8
1.3.3 Libraries	9
1.3.4 Stack, Heap and CPU statistics	9
1.4 Known Problems	9
1.5 Deployed Particle Sensors	9
2 Changelog	11
2.1 [Unreleased]	11
2.2 [0.4.3] - 2021-05-11	11
2.2.1 Fixed	11
2.2.2 Changed	11
2.3 [0.4.2] - 2021-02-25	11
2.3.1 Added	11
2.3.2 Fixed	11
2.4 [0.4.1] - 2021-01-07	12
2.4.1 Changed	12
2.4.2 Removed	12
2.5 [0.4.0] - 2020-12-03	12
2.5.1 Changed	12
2.6 [0.3.1] - 2020-11-30	12
2.6.1 Changed	12
2.6.2 Fixed	12
2.7 [0.3.0] - 2020-11-23	12
2.7.1 Added	12
2.7.2 Changed	12

2.8 [0.2.0] - 2020-11-19	13
2.8.1 Added	13
2.8.2 Changed	13
2.9 [0.1.0] - 2020-11-10	13
2.9.1 Added	13
2.9.2 Changed	13
2.9.3 Fixed	13
2.10 [0.0.1] - 2020-11-02	13
2.10.1 Added	13
3 Namespace Index	15
3.1 Namespace List	15
4 Hierarchical Index	17
4.1 Class Hierarchy	17
5 Class Index	19
5.1 Class List	19
6 File Index	21
6.1 File List	21
7 Namespace Documentation	23
7.1 utils Namespace Reference	23
7.1.1 Function Documentation	23
7.1.1.1 app_version()	23
7.1.1.2 calc_crc16()	23
7.1.1.3 calc_crc32()	24
7.1.1.4 get_formatted_time_string()	24
7.1.1.5 get_rand()	25
7.1.1.6 is_number()	25
7.1.1.7 string_to_bool()	25
7.1.1.8 to_hex_string()	26
8 Class Documentation	27
8.1 MeasureState::avg_data_s Struct Reference	27
8.1.1 Detailed Description	27
8.1.2 Member Function Documentation	27
8.1.2.1 operator+=()	27
8.1.2.2 operator/=()	28
8.1.3 Member Data Documentation	28
8.1.3.1 hum	28
8.1.3.2 particle_size	28
8.1.3.3 pm2_5	28

8.1.3.4 press	28
8.1.3.5 temp	28
8.2 Battery Class Reference	28
8.2.1 Detailed Description	29
8.2.2 Member Function Documentation	29
8.2.2.1 read_voltage()	29
8.3 SensorBME280::bme280_handle_t Struct Reference	29
8.3.1 Detailed Description	29
8.3.2 Member Data Documentation	29
8.3.2.1 bme280_cs	30
8.3.2.2 spi	30
8.4 BmeCommand Class Reference	30
8.4.1 Detailed Description	30
8.4.2 Constructor & Destructor Documentation	31
8.4.2.1 BmeCommand()	31
8.4.2.2 ~BmeCommand()	31
8.4.3 Member Function Documentation	31
8.4.3.1 execute_command()	31
8.4.4 Member Data Documentation	32
8.4.4.1 _bme280	32
8.5 Buttons Class Reference	32
8.5.1 Detailed Description	33
8.5.2 Constructor & Destructor Documentation	33
8.5.2.1 Buttons()	33
8.5.3 Member Function Documentation	33
8.5.3.1 button_left_pressed()	33
8.5.3.2 button_left_released()	33
8.5.3.3 button_right_pressed()	33
8.5.3.4 button_right_released()	34
8.5.3.5 is_left_pressed()	34
8.5.3.6 is_right_pressed()	34
8.5.4 Member Data Documentation	34
8.5.4.1 _button_left	34
8.5.4.2 _button_left_timer	34
8.5.4.3 _button_right	35
8.5.4.4 _button_right_timer	35
8.5.4.5 _event_flags	35
8.6 Command Class Reference	35
8.6.1 Detailed Description	36
8.6.2 Constructor & Destructor Documentation	36
8.6.2.1 Command()	36
8.6.2.2 ~Command()	37

8.6.3 Member Function Documentation	37
8.6.3.1 execute()	37
8.6.3.2 execute_command()	37
8.6.3.3 extract_parameters()	37
8.6.3.4 get_description()	38
8.6.3.5 get_name()	38
8.6.3.6 split_string()	38
8.6.4 Member Data Documentation	38
8.6.4.1 _description	39
8.6.4.2 _name	39
8.6.4.3 _usb_serial	39
8.7 CommandHandler Class Reference	39
8.7.1 Detailed Description	40
8.7.2 Constructor & Destructor Documentation	40
8.7.2.1 CommandHandler()	40
8.7.2.2 ~CommandHandler()	40
8.7.3 Member Function Documentation	40
8.7.3.1 connect()	41
8.7.3.2 disconnect()	41
8.7.3.3 find_command()	41
8.7.3.4 handle_usb_serial_command()	41
8.7.3.5 print_help()	41
8.7.3.6 print_prompt()	42
8.7.3.7 read_line_from_buffer()	42
8.7.3.8 separate_command_name()	42
8.7.3.9 serial_interrupt_handler()	42
8.7.4 Member Data Documentation	42
8.7.4.1 _buffer	43
8.7.4.2 _commands	43
8.7.4.3 _event_flags	43
8.7.4.4 _usb_serial	43
8.8 CommandState Class Reference	43
8.8.1 Detailed Description	44
8.8.2 Constructor & Destructor Documentation	44
8.8.2.1 CommandState()	44
8.8.2.2 ~CommandState()	44
8.8.3 Member Function Documentation	44
8.8.3.1 handle()	45
8.8.3.2 run()	45
8.8.4 Member Data Documentation	45
8.8.4.1 _command_handler	45
8.8.4.2 _event_flags	45

8.9 Communication Class Reference	46
8.9.1 Detailed Description	46
8.9.2 Member Enumeration Documentation	46
8.9.2.1 PacketType	46
8.9.3 Constructor & Destructor Documentation	47
8.9.3.1 Communication()	47
8.9.4 Member Function Documentation	47
8.9.4.1 authenticate()	47
8.9.4.2 run()	47
8.9.4.3 send_data()	47
8.9.4.4 send_measured_data()	48
8.9.4.5 send_status_data()	48
8.9.4.6 start()	48
8.9.4.7 stop()	48
8.9.5 Member Data Documentation	49
8.9.5.1 _data_mutex	49
8.9.5.2 _flags	49
8.9.5.3 _lora	49
8.9.5.4 _measured_data	49
8.9.5.5 _protocol	49
8.9.5.6 _status_update_ticker	49
8.9.5.7 _thread	50
8.10 Config Class Reference	50
8.10.1 Detailed Description	51
8.10.2 Member Enumeration Documentation	51
8.10.2.1 ConfigType	51
8.10.3 Member Function Documentation	51
8.10.3.1 get()	51
8.10.3.2 get_entry()	52
8.10.3.3 read_config_from_flash()	52
8.10.3.4 read_from_flash()	52
8.10.3.5 reset_config()	53
8.10.3.6 save_default_config()	53
8.10.3.7 set_value()	53
8.10.3.8 write_config_to_flash()	54
8.10.3.9 write_to_flash()	54
8.10.4 Member Data Documentation	54
8.10.4.1 default_config	54
8.10.4.2 global_config	55
8.11 Config::config_value_t Union Reference	55
8.11.1 Detailed Description	55
8.11.2 Member Data Documentation	55

8.11.2.1 bool_v	55
8.11.2.2 float_v	55
8.11.2.3 uint32_v	56
8.12 ConfigCommand Class Reference	56
8.12.1 Detailed Description	56
8.12.2 Constructor & Destructor Documentation	56
8.12.2.1 ConfigCommand()	56
8.12.2.2 ~ConfigCommand()	57
8.12.3 Member Function Documentation	57
8.12.3.1 execute_command()	57
8.12.3.2 handle_parameters()	57
8.12.3.3 print_current_config()	58
8.12.4 Member Data Documentation	58
8.12.4.1 _flash	58
8.13 Config::entry_t Struct Reference	58
8.13.1 Detailed Description	58
8.13.2 Member Data Documentation	58
8.13.2.1 description	59
8.13.2.2 max	59
8.13.2.3 min	59
8.13.2.4 name	59
8.13.2.5 type	59
8.13.2.6 value	59
8.14 ErrorState Class Reference	59
8.14.1 Detailed Description	60
8.14.2 Constructor & Destructor Documentation	60
8.14.2.1 ErrorState()	60
8.14.3 Member Function Documentation	60
8.14.3.1 handle()	60
8.14.4 Member Data Documentation	61
8.14.4.1 _control_event_flags	61
8.15 InfoCommand Class Reference	61
8.15.1 Detailed Description	61
8.15.2 Constructor & Destructor Documentation	61
8.15.2.1 InfoCommand()	61
8.15.2.2 ~InfoCommand()	62
8.15.3 Member Function Documentation	62
8.15.3.1 execute_command()	62
8.16 InitState Class Reference	62
8.16.1 Detailed Description	63
8.16.2 Constructor & Destructor Documentation	63
8.16.2.1 InitState()	63

8.16.2.2 ~InitState()	64
8.16.3 Member Function Documentation	64
8.16.3.1 bme280_test()	64
8.16.3.2 handle()	64
8.16.3.3 init()	65
8.16.3.4 init_rtc()	65
8.16.3.5 sps30_test()	65
8.16.4 Member Data Documentation	65
8.16.4.1 _bme280	65
8.16.4.2 _buttons	65
8.16.4.3 _communication	66
8.16.4.4 _flash	66
8.16.4.5 _sd_card	66
8.16.4.6 _sps30	66
8.17 LedIndicator Class Reference	66
8.17.1 Detailed Description	67
8.17.2 Member Enumeration Documentation	67
8.17.2.1 Color	67
8.17.3 Constructor & Destructor Documentation	68
8.17.3.1 LedIndicator()	68
8.17.3.2 ~LedIndicator()	68
8.17.4 Member Function Documentation	68
8.17.4.1 display_battery_level_for()	68
8.17.4.2 display_status_for()	68
8.17.4.3 switch_battery_leds()	69
8.17.4.4 switch_status_leds()	69
8.17.4.5 turn_off_battery_leds()	69
8.17.4.6 turn_off_status_leds()	69
8.17.5 Member Data Documentation	69
8.17.5.1 _bat_led_1	70
8.17.5.2 _bat_led_2	70
8.17.5.3 _bat_led_3	70
8.17.5.4 _bat_led_timeout	70
8.17.5.5 _status_led_green	70
8.17.5.6 _status_led_red	70
8.17.5.7 _status_led_timeout	70
8.18 Logger Class Reference	71
8.18.1 Detailed Description	71
8.18.2 Constructor & Destructor Documentation	71
8.18.2.1 Logger() [1/2]	71
8.18.2.2 ~Logger()	72
8.18.2.3 Logger() [2/2]	72

8.18.3 Member Function Documentation	72
8.18.3.1 compose_log_line()	72
8.18.3.2 init()	72
8.18.3.3 log_internal()	73
8.18.3.4 operator=()	73
8.18.4 Member Data Documentation	73
8.18.4.1 _file_name	73
8.18.4.2 _initialized	73
8.18.4.3 _sd_card	74
8.19 Lora Class Reference	74
8.19.1 Detailed Description	75
8.19.2 Constructor & Destructor Documentation	75
8.19.2.1 Lora()	75
8.19.3 Member Function Documentation	75
8.19.3.1 cad_done_cb()	75
8.19.3.2 fhss_change_channel_cb()	75
8.19.3.3 init()	76
8.19.3.4 receive()	76
8.19.3.5 rx_done_cb()	76
8.19.3.6 rx_error_cb()	76
8.19.3.7 rx_timeout_cb()	77
8.19.3.8 send()	77
8.19.3.9 set_rx_config()	77
8.19.3.10 set_tx_config()	77
8.19.3.11 sleep()	77
8.19.3.12 tx_done_cb()	77
8.19.3.13 tx_timeout_cb()	78
8.19.4 Member Data Documentation	78
8.19.4.1 _event_flags	78
8.19.4.2 _radio	78
8.19.4.3 _radio_callbacks	78
8.19.4.4 _rx_timeout	78
8.19.4.5 rx_payload	78
8.19.4.6 rx_payload_len	78
8.19.4.7 rx_rssi	79
8.19.4.8 rx_snr	79
8.20 MeasureState Class Reference	79
8.20.1 Detailed Description	80
8.20.2 Member Typedef Documentation	80
8.20.2.1 avg_data_t	80
8.20.3 Member Enumeration Documentation	80
8.20.3.1 SPS30State	80

8.20.4 Constructor & Destructor Documentation	81
8.20.4.1 MeasureState()	81
8.20.4.2 ~MeasureState()	81
8.20.5 Member Function Documentation	81
8.20.5.1 get_sps30_info()	82
8.20.5.2 handle()	82
8.20.5.3 init_measurement_file()	82
8.20.5.4 run()	82
8.20.5.5 send_measurement_data()	83
8.20.5.6 set_create_new_file_flag()	83
8.20.5.7 set_do_measurement_flag()	83
8.20.5.8 set_sps30_startup_done_flag()	83
8.20.5.9 store_measurement_data()	83
8.20.6 Member Data Documentation	84
8.20.6.1 _avg_data	84
8.20.6.2 _bme280	84
8.20.6.3 _communication	84
8.20.6.4 _data_file_name	84
8.20.6.5 _event_flags	84
8.20.6.6 _measurement_ticker	84
8.20.6.7 _new_file_ticker	85
8.20.6.8 _sd_card	85
8.20.6.9 _sps30	85
8.20.6.10 _sps30_data_buffer	85
8.20.6.11 _sps30_info	85
8.20.6.12 _sps30_startup_timeout	85
8.20.6.13 _sps30_state	85
8.21 Protocol Class Reference	86
8.21.1 Detailed Description	87
8.21.2 Constructor & Destructor Documentation	87
8.21.2.1 Protocol()	87
8.21.3 Member Function Documentation	87
8.21.3.1 __attribute__([1/7])	87
8.21.3.2 __attribute__([2/7])	87
8.21.3.3 __attribute__([3/7])	87
8.21.3.4 __attribute__([4/7])	88
8.21.3.5 __attribute__([5/7])	88
8.21.3.6 __attribute__([6/7])	88
8.21.3.7 __attribute__([7/7])	88
8.21.3.8 auth_reset()	88
8.21.3.9 check_ack()	88
8.21.3.10 check_auth()	89

8.21.3.11 <code>crc_check()</code>	89
8.21.3.12 <code>get_hw_uuid()</code>	89
8.21.3.13 <code>get_sw_uuid()</code>	90
8.21.3.14 <code>is_auth_done()</code>	90
8.21.3.15 <code>read_received()</code>	90
8.21.3.16 <code>send_auth()</code>	90
8.21.3.17 <code>send_measured_data()</code>	90
8.21.3.18 <code>send_status_data()</code>	91
8.21.4 Member Data Documentation	91
8.21.4.1 <code>_base_crc</code>	91
8.21.4.2 <code>_client_auth_done</code>	91
8.21.4.3 <code>_lora</code>	91
8.21.4.4 <code>_node_crc</code>	91
8.21.4.5 <code>_node_id</code>	92
8.21.4.6 <code>base_ack_t</code>	92
8.21.4.7 <code>base_auth_t</code>	92
8.21.4.8 <code>measured_data_t</code>	92
8.21.4.9 <code>node_auth_t</code>	92
8.21.4.10 <code>node_measured_data_t</code>	92
8.21.4.11 <code>node_status_data_t</code>	92
8.21.4.12 <code>status_data_t</code>	93
8.22 SDCard Class Reference	93
8.22.1 Detailed Description	93
8.22.2 Constructor & Destructor Documentation	94
8.22.2.1 <code>SDCard()</code>	94
8.22.3 Member Function Documentation	94
8.22.3.1 <code>check_available_memory()</code>	94
8.22.3.2 <code>deinit()</code>	94
8.22.3.3 <code>format()</code>	95
8.22.3.4 <code>init()</code>	95
8.22.3.5 <code>is_full()</code>	95
8.22.3.6 <code>is_inserted()</code>	95
8.22.3.7 <code>write()</code>	95
8.22.4 Member Data Documentation	96
8.22.4.1 <code>_fat_file_system</code>	96
8.22.4.2 <code>_initialized</code>	96
8.22.4.3 <code>_is_full</code>	96
8.22.4.4 <code>_n_detect</code>	96
8.22.4.5 <code>_sd_block_device</code>	96
8.22.4.6 <code>_sd_card_mutex</code>	97
8.23 SensorBME280 Class Reference	97
8.23.1 Detailed Description	98

8.23.2 Member Typedef Documentation	98
8.23.2.1 measurement_t	98
8.23.3 Constructor & Destructor Documentation	98
8.23.3.1 SensorBME280()	98
8.23.3.2 ~SensorBME280()	98
8.23.4 Member Function Documentation	98
8.23.4.1 bme280_delay_us()	98
8.23.4.2 bme280_spi_init()	99
8.23.4.3 bme280_spi_read()	99
8.23.4.4 bme280_spi_write()	99
8.23.4.5 correct_data()	100
8.23.4.6 init()	100
8.23.4.7 read_measurement()	100
8.23.4.8 read_sensor_data()	101
8.23.5 Member Data Documentation	101
8.23.5.1 _bme280_cs	101
8.23.5.2 _bme280_handle	101
8.23.5.3 _dev	101
8.23.5.4 _spi	102
8.24 SensorSPS30 Class Reference	102
8.24.1 Detailed Description	103
8.24.2 Member Typedef Documentation	103
8.24.2.1 measurement_t	103
8.24.3 Constructor & Destructor Documentation	103
8.24.3.1 SensorSPS30()	103
8.24.3.2 ~SensorSPS30()	103
8.24.4 Member Function Documentation	103
8.24.4.1 correct_data()	103
8.24.4.2 get_serial()	104
8.24.4.3 power_off()	104
8.24.4.4 power_on()	104
8.24.4.5 probe()	104
8.24.4.6 read_measurement()	104
8.24.4.7 read_version()	105
8.24.4.8 start_manual_fan_cleaning()	105
8.24.4.9 start_measurement()	105
8.24.4.10 stop_measurement()	106
8.24.5 Member Data Documentation	106
8.24.5.1 _serial	106
8.24.5.2 _supply_enable	106
8.25 SpsCommand Class Reference	106
8.25.1 Detailed Description	107

8.25.2 Constructor & Destructor Documentation	107
8.25.2.1 SpsCommand()	107
8.25.2.2 ~SpsCommand()	107
8.25.3 Member Function Documentation	107
8.25.3.1 execute_command()	108
8.25.4 Member Data Documentation	108
8.25.4.1 _sps30	108
8.26 State Class Reference	108
8.26.1 Detailed Description	109
8.26.2 Member Enumeration Documentation	109
8.26.2.1 Result	109
8.26.3 Constructor & Destructor Documentation	109
8.26.3.1 State()	109
8.26.3.2 ~State()	110
8.26.4 Member Function Documentation	110
8.26.4.1 handle()	110
8.26.5 Member Data Documentation	110
8.26.5.1 _led_indicator	110
8.27 StateContext Class Reference	111
8.27.1 Detailed Description	111
8.27.2 Constructor & Destructor Documentation	111
8.27.2.1 StateContext()	111
8.27.2.2 ~StateContext()	112
8.27.3 Member Function Documentation	112
8.27.3.1 next_state()	112
8.27.3.2 run()	112
8.27.4 Member Data Documentation	112
8.27.4.1 _command_state	113
8.27.4.2 _current_state	113
8.27.4.3 _error_state	113
8.27.4.4 _init_state	113
8.27.4.5 _measure_state	113
8.28 TimeCommand Class Reference	113
8.28.1 Detailed Description	114
8.28.2 Constructor & Destructor Documentation	114
8.28.2.1 TimeCommand()	114
8.28.2.2 ~TimeCommand()	115
8.28.3 Member Function Documentation	115
8.28.3.1 calibrate()	115
8.28.3.2 execute_command()	115
8.28.3.3 parse_datetime()	115
8.28.3.4 wait_for_second_toggle()	116

8.28.4 Member Data Documentation	116
8.28.4.1 _cal_time_sec	116
8.29 version_number_s Struct Reference	116
8.29.1 Member Data Documentation	116
8.29.1.1 build	116
8.29.1.2 major	117
8.29.1.3 minor	117
9 File Documentation	119
9.1 CHANGELOG.md File Reference	119
9.2 README.md File Reference	119
9.3 src/command/BmeCommand.cpp File Reference	119
9.4 src/command/BmeCommand.h File Reference	119
9.5 src/command/Command.cpp File Reference	119
9.6 src/command/Command.h File Reference	120
9.7 src/command/CommandHandler.cpp File Reference	120
9.8 src/command/CommandHandler.h File Reference	120
9.9 src/command/ConfigCommand.cpp File Reference	120
9.10 src/command/ConfigCommand.h File Reference	121
9.11 src/command/InfoCommand.cpp File Reference	121
9.12 src/command/InfoCommand.h File Reference	121
9.13 src/command/SpsCommand.cpp File Reference	121
9.14 src/command/SpsCommand.h File Reference	121
9.15 src/command/TimeCommand.cpp File Reference	122
9.15.1 Macro Definition Documentation	122
9.15.1.1 MS_PER_DAY	122
9.16 src/command/TimeCommand.h File Reference	122
9.17 src/communication/Communication.cpp File Reference	122
9.17.1 Macro Definition Documentation	123
9.17.1.1 FLAG_SEND_MEASURED_DATA	123
9.17.1.2 FLAG_SEND_STATUS_DATA	123
9.18 src/communication/Communication.h File Reference	123
9.19 src/communication/Protocol.cpp File Reference	123
9.19.1 Macro Definition Documentation	124
9.19.1.1 MEASURED_DATA_TYPE_ID	124
9.19.1.2 SMARTMOTE_ACK	124
9.19.1.3 SMARTMOTE_AUTH_ID	124
9.19.1.4 SMARTMOTE_HW_ID	124
9.19.1.5 STATUS_DATA_TYPE_ID	124
9.20 src/communication/Protocol.h File Reference	124
9.21 src/config/Config.cpp File Reference	125
9.22 src/config/Config.h File Reference	125

9.22.1 Macro Definition Documentation	125
9.22.1.1 NUM_CONFIG_ENTRIES	125
9.23 src/defs/defines.h File Reference	125
9.23.1 Macro Definition Documentation	127
9.23.1.1 BME280_TEST_STARTUP_TIME_MS	127
9.23.1.2 BUTTON_COMMAND_MIN_PRESS_TIME	127
9.23.1.3 BUTTON_DEBOUNCE_TIME	127
9.23.1.4 COMMAND_HANDLER_MAX_BUF_SIZE	127
9.23.1.5 DATA_FILE_CREATION_PERIOD_SEC	128
9.23.1.6 FLAG_BUTTON_COMMAND	128
9.23.1.7 FLAG_BUTTON_MEASURE	128
9.23.1.8 FLAG_BUTTON_STATUS	128
9.23.1.9 FLAG_CREATE_NEW_FILE	128
9.23.1.10 FLAG_DO_MEASUREMENT	128
9.23.1.11 FLAG_LORA_RX_DONE	128
9.23.1.12 FLAG_LORA_RX_TIMEOUT	129
9.23.1.13 FLAG_LORA_TX_DONE	129
9.23.1.14 FLAG_RECEIVED_COMMAND	129
9.23.1.15 FLAG_SPS30_STARTUP_DONE	129
9.23.1.16 LORA_RESEND_WAIT_MS	129
9.23.1.17 LORA_RX_TIMEOUT_MS	129
9.23.1.18 MAX_SEND_DATA_ATTEMPTS	130
9.23.1.19 STORAGE_ADDR_CONFIG	130
9.23.1.20 VERSION_BUILD	130
9.23.1.21 VERSION_MAJOR	130
9.23.1.22 VERSION_MINOR	130
9.23.2 Enumeration Type Documentation	130
9.23.2.1 LogLevel	130
9.23.3 Variable Documentation	131
9.23.3.1 bme_command_description	131
9.23.3.2 bme_command_name	131
9.23.3.3 config_command_description	131
9.23.3.4 config_command_name	131
9.23.3.5 info_command_description	132
9.23.3.6 info_command_name	132
9.23.3.7 log_level_names	132
9.23.3.8 sps_command_description	132
9.23.3.9 sps_command_name	132
9.23.3.10 time_command_description	133
9.23.3.11 time_command_name	133
9.23.3.12 version	133
9.24 src/driver/Battery.cpp File Reference	133

9.24.1 Function Documentation	133
9.24.1.1 analogin_init_direct()	134
9.24.1.2 bat_level_en()	134
9.24.1.3 bat_level_input()	134
9.25 src/driver/Battery.h File Reference	134
9.26 src/driver/Buttons.cpp File Reference	134
9.27 src/driver/Buttons.h File Reference	134
9.28 src/driver/LedIndicator.cpp File Reference	135
9.28.1 Macro Definition Documentation	135
9.28.1.1 BATTERY_THRESHOLD_LED_1	135
9.28.1.2 BATTERY_THRESHOLD_LED_2	135
9.28.1.3 BATTERY_THRESHOLD_LED_3	135
9.29 src/driver/LedIndicator.h File Reference	135
9.30 src/driver/Lora.cpp File Reference	136
9.31 src/driver/Lora.h File Reference	136
9.31.1 Macro Definition Documentation	136
9.31.1.1 LORA_CFG_MODEM	136
9.31.1.2 LORA_CFG_RX_BANDWIDTH_AFC	136
9.31.1.3 LORA_CFG_RX_PAYLOAD_LEN	137
9.31.1.4 LORA_CFG_TX_FDEV	137
9.32 src/driver/SDCard.cpp File Reference	137
9.33 src/driver/SDCard.h File Reference	137
9.34 src/driver/SensorBME280.cpp File Reference	137
9.35 src/driver/SensorBME280.h File Reference	137
9.36 src/driver/SensorSPS30.cpp File Reference	138
9.37 src/driver/SensorSPS30.h File Reference	138
9.38 src/logging/Logger.cpp File Reference	138
9.39 src/logging/Logger.h File Reference	138
9.39.1 Macro Definition Documentation	139
9.39.1.1 __FILENAME__	139
9.39.1.2 LOG	139
9.40 src/main.cpp File Reference	139
9.40.1 Function Documentation	140
9.40.1.1 main()	140
9.41 src/state/CommandState.cpp File Reference	140
9.42 src/state/CommandState.h File Reference	140
9.43 src/state/ErrorState.cpp File Reference	140
9.43.1 Function Documentation	140
9.43.1.1 mbed_error_reboot_callback()	141
9.44 src/state/ErrorState.h File Reference	141
9.45 src/state/InitState.cpp File Reference	141
9.45.1 Macro Definition Documentation	141

9.45.1.1	TIMESTAMP_20200101	141
9.46	src/state/InitState.h File Reference	141
9.47	src/state/MeasureState.cpp File Reference	142
9.48	src/state/MeasureState.h File Reference	142
9.49	src/state/State.cpp File Reference	142
9.50	src/state/State.h File Reference	142
9.51	src/state/StateContext.cpp File Reference	143
9.52	src/state/StateContext.h File Reference	143
9.53	src/utls/utls.cpp File Reference	143
9.54	src/utls/utls.h File Reference	144
9.54.1	Macro Definition Documentation	144
9.54.1.1	REVERSE_2_BYTE	145
9.54.1.2	TRY	145
Index		147

Chapter 1

SensMat Particle Sensor <!-- omit in toc -->

- Description
 - Features
 - Documentation
 - Program Flow Chart
- Usage
 - Interfaces
 - Starting the sensor
 - Charging the sensor
 - SD card logging
 - USB serial command interface
 - * Supported commands
 - help
 - time
 - config
 - sps
 - bme
 - info
 - * RTC Calibration and Settings
- Development
 - Preparations
 - Toolchain
 - * Option 1: mbed-cli (tested on Linux)
 - Building
 - Flashing
 - * Option 2: SW4STM32 (Eclipse with build tools, tested on Windows)
 - Libraries
- Known Problems
- Deployed Particle Sensors

1.1 Description

This is the documentation for the SensMat Particle Sensor software, which runs on the STM32L476RGT6 micro-controller.

1.1.1 Features

- Particulate matter measurement using the `Sensirion SensorSPS30` sensor
- Ambient temperature, relative humidity and pressure measurement using the `Bosch Sensortec SensorBME280` sensor
- `Battery` operated (operation time depending on measurement frequency: More than 2 months at a measurement interval of 15 minutes)
- On-board battery charger
- Logging of measurement data to micro SD card
- LoRa module to upload data via Smartmote gateway to SensMat cloud
- USB serial interface for configuration and calibration

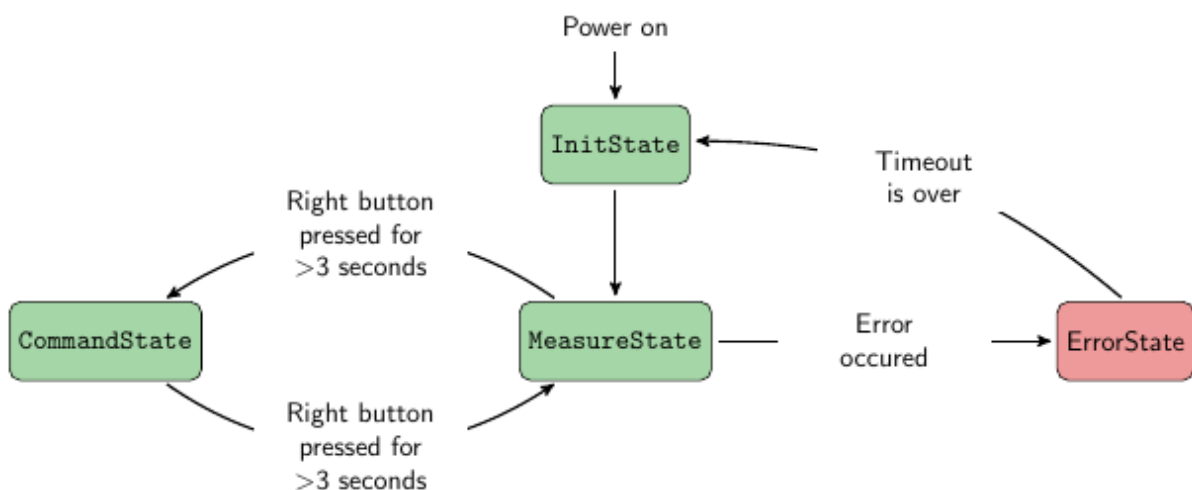
1.1.2 Documentation

The source code documentation can be built with `doxygen` by executing `doxygen Doxyfile` in the folder `SensMat/Software/Particle_Sensor`. This will create the html documentation in the folder `SensMat/Software/Particle_Sensor/doc/html/`.

To create a PDF file from the LaTeX sources the program `pdflatex` is needed. Then execute `make` in `SensMat/Software/Particle_Sensor/doc/latex/` which compiles the documentation to a PDF named `refman.pdf`.

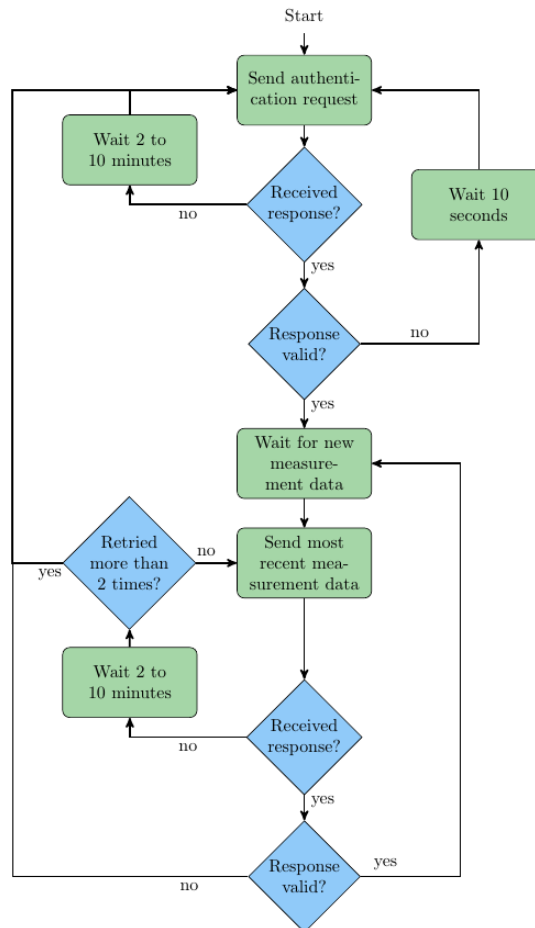
1.1.3 Program Flow Chart

The application uses two threads. The main thread implements a simple state machine with four states and the following transitions:





In a second thread, which is started in the `InitState` of the main thread, the LoRa communication is handled according to the protocol specification from SensMat:

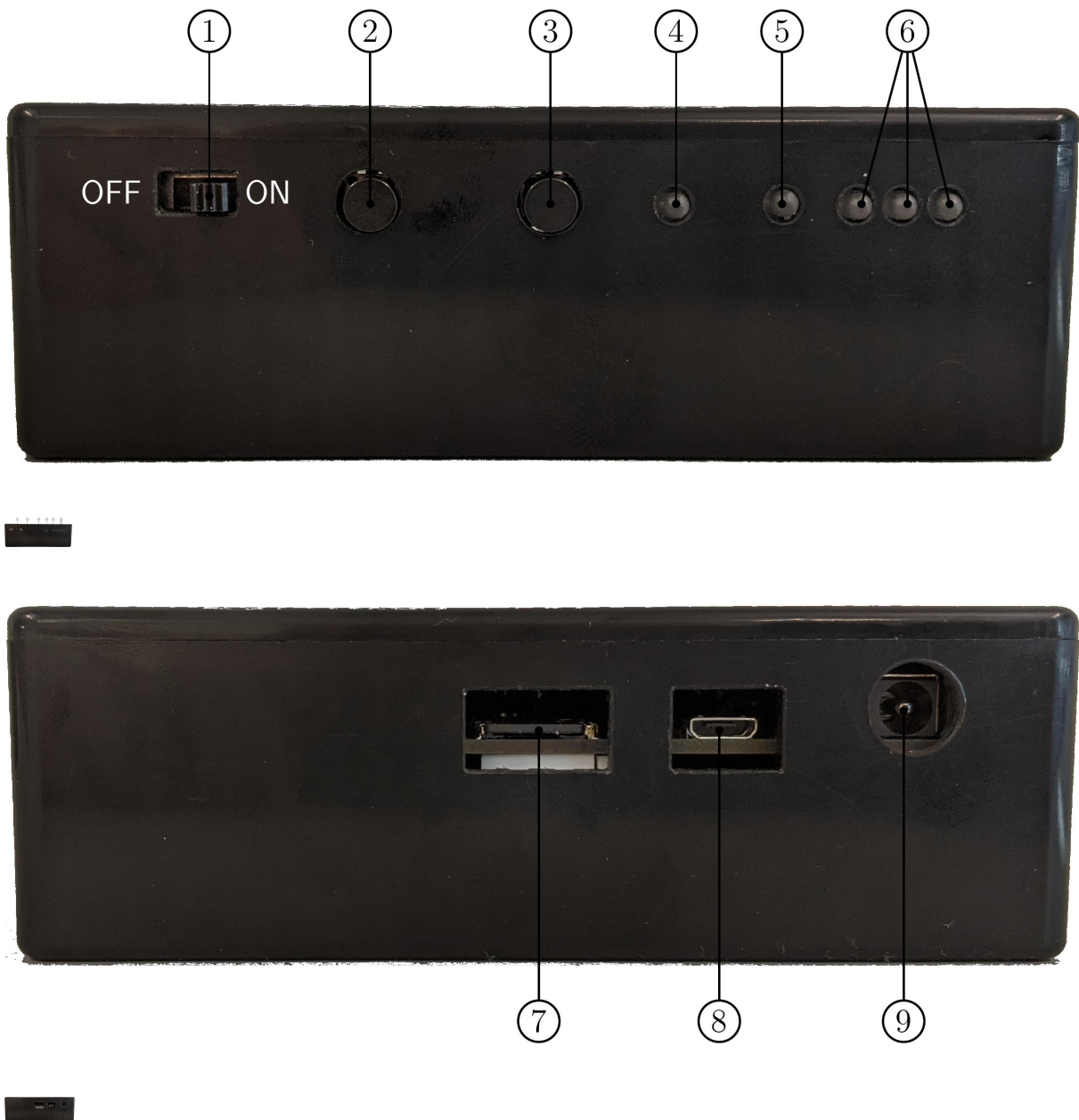


1.2 Usage

NOTE:

The device is not waterproof and should therefore only be placed indoors or at a weatherproof location. Placing it in direct sunlight can lead to measurement deviations due to the black enclosure.

1.2.1 Interfaces



1. **Power switch** to turn the power supply on and off
2. **Button** to manually start a single measurement
3. **Button** to display the status and battery charge level for 3 seconds
4. **Status LED** indicating the status of the program (green => no error, red => error)
5. **Charge LED** which indicates if the battery is charging or not
6. **Battery level LEDs** indicating the battery status
7. **SD card slot** for a micro SD card to store measurement data and log data
8. **Micro USB connector** to configure the particle sensor using a PC
9. **Power connector** to connect the power adapter (9-22V) to for battery charging

1.2.2 Starting the sensor

1. Make sure that the gateway is running.
2. Switch on the power supply by setting the power switch to ON (to the right).
3. The status LED lights up orange for a while during the start-up process.
4. As soon as the status LED turns green for 3 seconds, the sensor starts reporting measurement data.
5. Optional: By pressing the right button, the status of the sensor (status LED: green => no error; red => error) and the battery level (all battery LEDs green => full; all LEDs off => low/empty) are shown for 3 seconds.
6. Optional: By pressing the left button, a single measurement can be started.

1.2.3 Charging the sensor

1. Switch off the power by setting the power switch to OFF (to the left).
2. Connect the power adapter to the power connector. The charging process will start immediately.
3. While charging, the charge LED is orange. The charging time is approximately 3 hours.
4. As soon as the charge LED turns off, the battery is fully charged and the power adapter can be unplugged and the sensor can be started again.

1.2.4 SD card logging

If logging of measurement data is enabled in the configuration, the raw measurements are stored as CSV file on the microSD card. Every 24 hours, a new file is created.

At each start of the particle sensor, a log file is created which logs events from the application according to the configured log level. This file should help to debug the firmware.

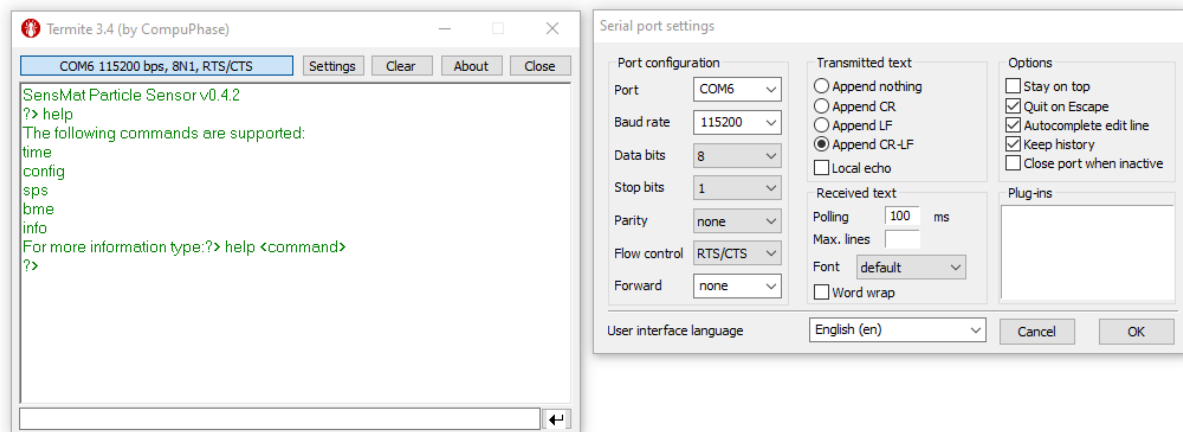
1.2.5 USB serial command interface

The USB interface can be used to configure several parameters of the particle sensor, to set and calibrate the RTC, to read version numbers or to test the sensors.

To access it, the following steps are necessary:

1. Turn on the particle sensor and wait until the status LED is green or off.
2. Press the right button (status button) for more than 3 seconds.
3. After releasing the button, the status LED is orange, indicating that the particle sensor is in the command state.
4. Connect the particle sensor with a micro USB cable to a PC.
5. Open a serial terminal (e.g. [Termite](#)) at the PC and connect to the right port. The settings for the serial connection are visible in the screenshot below these steps.
6. You should receive something like `SensMat Particle Sensor v0.4.2` and in the following line `?>`, as shown in the figure below.

7. Send `help` to see the supported commands and `help <command name>` to see the usage information for a specific command.
8. To conveniently execute multiple commands in a sequence, the python script `scripts/configure.py` may help.



1.2.5.1 Supported commands

In the following, all supported commands are listed, including a description how to use them.

1.2.5.1.1 help This command prints all supported commands in a list. To view the command description for a specific command, type: `help <command name>`. E.g. for the `time`-command: `help time`

1.2.5.1.2 time Usage: `time [OPTIONS]`

Options:

<code><none></code>	Print current time
<code>set YYYY-MM-DD HH:MM:SS</code>	Set time in specified format
<code>tick <num_ticks></code>	Print for <code><num_ticks></code> seconds every time, the RTC second counter toggles, the amount of passed seconds. Used to measure RTC time drift.
<code>calibrate <milliseconds></code>	Calibrate the RTC by sending the time drift per day in milliseconds

1.2.5.1.3 config Usage: `config [OPTIONS]`

Options:

<code><none></code>	Print current configuration
<code><name>=<value></code>	Set the configuration parameter <code><name></code> to the given <code><value></code> . Multiple parameters can be set in one command by separating key-value-pairs with spaces.

1.2.5.1.4 sps Usage: `sps COMMAND`

Commands:

<code>on</code>	Enable SPS30 power supply
<code>off</code>	Disable SPS30 power supply
<code>probe</code>	Perform probe, to check if SPS30 is responding
<code>serial</code>	Print serial number
<code>version</code>	Print version numbers for firmware, SHDLIC and hardware
<code>start</code>	Start measurement procedure
<code>stop</code>	Stop measurement procedure
<code>read</code>	Read measurement values from sensor
<code>clean</code>	Start fan cleaning (only works after "sps start")

1.2.5.1.5 bme Usage: `bme COMMAND`

Commands:

<code>init</code>	Initialize SensorBME280
<code>read</code>	Read measurement values from sensor

1.2.5.1.6 info Usage: info
Print software version and UUIDs

1.2.5.2 RTC Calibration and Settings

The time can be set manually via the USB serial interface using the `time` command.

But it is more convenient to do it by using the python script `scripts/rtc_tool.py`, which can set the RTC to the current date and time. `rtc_tool.py` can also calibrate the RTC to be more accurate.

1.3 Development

The software is based on [Arm Mbed OS 6](#), which is a RTOS for embedded systems and it includes many drivers and stacks.

For building and configurint this Arm Mbed OS project, the code management tool `mbed-cli` is used.

1.3.1 Preparations

- For this project, `git` is used as version control system (VCS), which needs to be installed.
- To use the software, clone the SensMat repository from `git.tugraz.at`:
`git clone ssh://git@git.tugraz.at/SensMat.git`
- Change into the directory of the Particle Sensor Software:
`cd SensMat/Software/Particle_Sensor`
- Since `mbed-os` and other libraries are included as git submodules in the repository, it is necessary to download (update) the submodules:
`git submodule update --init --recursive`
- Check if the Arm Mbed OS version in the submodule `mbed-os` is 6.4.0 or higher by typing:
`git submodule` which will print something like:
`8ef0a435b2356f8159dea8e427b2935d177309f8 mbed-os (mbed-os-6.4.0)`
- If another version is shown, check out version 6.4.0 with `git checkout mbed-os-6.4.0` in the folder `Software/Framework/mbed-os`
- In the projects root folder, a git patch (`mbed-os.patch`) for the `mbed-os` submodule is included, which should be applied with `git apply ../../Particle_Sensor/mbed-os.patch` in the folder `Software/Framework/mbed-os`. This patch applies the following changes: 1) It increases the default stack size for the LoRa driver thread from 1024 bytes to 4096 in file `SX126X_LoRaRadio.cpp` line 111. This is necessary, because 1024 is sometimes too little, leading to a crash of the firmware. 2) Add a `.mbedignore` file to the `mbed-os` folder to ignore unused modules when compiling with the `mbed` command. This leads to a faster build process.

1.3.2 Toolchain

There are multiple options how to develop build and flash the firmware. In the following two possible toolchain options are explained.

1.3.2.1 Option 1: mbed-cli (tested on Linux)

Needed tools:

- `mbed-cli`: The build system for mbed OS projects.
- `GNU Arm Embedded Toolchain`
- (Optional) `stlink`: Tool to flash ST microcontrollers
- `openocd` or `pyocd` (only for debugging)

Setting the toolchain path is done with:

```
mbed config -G GCC_ARM_PATH /usr/bin/arm-none-eabi-gcc
```

1.3.2.1.1 Building To build the project for the particle sensor hardware for PCB version 2 using the release profile, execute:

```
mbed compile --profile release --source . --source ../Framework/mbed-os
```

It could be necessary to install further python dependencies with:

```
sudo pip3 install -r ../Framework/mbed-os/requirements.txt
```

To change the toolchain, the toolchain path or the target, edit the `.mbed` file.

1.3.2.1.2 Flashing When using a Nucleo board to flash the firmware, just copy the created binary file `BUILD/PART_SENS_V3/GCC_ARM-RELEASE/Particle_Sensor.bin` to the storage device emulated by the Nucleo board.

To flash the binary with `stlink` use:

```
st-flash write ./BUILD/PART_SENS_V3/GCC_ARM-RELEASE/Particle_Sensor.bin
0x8000000
```

1.3.2.2 Option 2: SW4STM32 (Eclipse with build tools, tested on Windows)

With the `mbed-cli` tool, a Makefile was exported which allows to compile the project with the `make` tool or with an Eclipse IDE with C/C++ support and the necessary GNU ARM Embedded toolchain installed. System Workbench for STM32 (SW4STM32) is a version of eclipse with all necessary tools included.

NOTE:

The Makefile includes the configuration for the release build. For a debug build with debug symbols, the Makefile needs to be adapted.

- Install `SW4STM32`
- Open SW4STM32 and open project folder with `File -> Open Projects from File System...` and select folder `Particle_Sensor` from the SensMat repository.
- Maybe it is necessary to add the locations of the `make` and the GNU ARM Embedded toolchain to the system path.
- To build the project, press the hammer symbol
- To debug the project, press the arrow next to the debug symbol (bug) and select `Debug Configurations...`
- In this window, uncollapse the `Ac STM32 Debugging configurations` and select the configuration `Particle_Sensor Debug`
- Connect the debugger (e.g. the one included on a Nucleo board) to the Particle Sensor 6-pin SWD header. With a USB cable connect the debugger to the PC.
- Then press the `Debug` button. This will build the binary, if not already done, flash the binary, start the debugger and switch eclipse into the debug perspective.

1.3.3 Libraries

For the two sensors, the manufacturers provide drivers, which were included in the `lib` folder of the project. For reference, this are the git repositories, where the libraries are from:

- [Sensirion SensorSPS30 UART driver](#)
- [Bosch Sensortec SensorBME280 driver](#)

1.3.4 Stack, Heap and CPU statistics

If `PRINT_STATS` is defined, statistics about stack, heap and CPU usage is printed via the UART interface if the status button is pressed. This can be used to debug power management (deep sleep/sleep/run durations) or memory leakage problems.

NOTE:

The microcontroller only goes in deep sleep mode if the firmware is compiled with the **release** profile.

1.4 Known Problems

- [Communication](#) over the USB serial interface does not work reliably when compiling in debug profile (at least on Linux). This is caused by using a while-loop in the receive interrupt handler instead of an if-statement when polling for available characters. But if an if-statement is used, the USB serial interface does not work on Windows.
- If the microcontroller is in deep sleep mode when it is flashed with a new firmware using st-flash, it only erases the flash memory but writing the new firmware will fail. Just reconnect the debugger from the PC and flash again. Another solution is to switch sensor node into command state, as in this state the microcontroller is not in deep sleep mode.

1.5 Deployed Particle Sensors

PCB #	HWUUID	SPS30 Serial #
1	0x274784d6	15E099066D0D420B
2	0xbe90f3bf	6F5BAC226EB9E8CE
3	0xb2881e43	AFD8C4D630E0E33E
4	0x73eba155	FD1922CB58117D02
5	0xbd69061e	00941E593F1072DE
6	0x7e8af484	FF3297200FEEBA54
7	0x92b16a1b	C498FD7969A0A47B
8	0x95444f80	244063AEB5C5B31E
9	0xbb1a5576	7D6E454FFCFF64BB
10	0x5b392615	C903D98B57446B25
11	0x9442d23b	4484E7F7A263A28F
12	0xe7d0d0f1	AB41F5B2330B0ECB
13	0xb208f50b	BA5D484176C95811
14	0xe6d64d4a	934B87913FD5D3F2
15	0xc09c6a7a	E9DEFAD71B7FAF3F

PCB #	HWUUID	SPS30 Serial #
16	0xc59627fb	9BC4514C922B18F5
17	0xe0da3bb4	98F9290B17B584E9
18	0x9744e944	DE685F7756526A66
19	0x0ee103ef	3D0EBA6AC17D82AF
20	0x7c759c9e	E7D536CB98CF8EA6
21	0x640cfe8a	6077E177E4EB78CC
22	0x2de283d9	07D563C944A1CE29
23	0x35316342	CEDFFF0110924553
24	0xe56c526c	8742721F9C4035F5

Chapter 2

Changelog

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#), and this project adheres to [Semantic Versioning](#).

The version number of the program is defined in [src/defs/defines.h](#) and needs to be adjusted according to the version in this file.

2.1 [Unreleased]

2.2 [0.4.3] - 2021-05-11

2.2.1 Fixed

- Rename class BME280 to [SensorBME280](#) and SPS30 to [SensorSPS30](#) due to name conflicts on Windows

2.2.2 Changed

- Update usage description for USB serial interface commands.

2.3 [0.4.2] - 2021-02-25

2.3.1 Added

- Compute moving average over a configurable number of measurement values

2.3.2 Fixed

- Increase buffer size for measurement values written to SD card.
- Retry reading battery level maximum 5 times if it fails.

2.4 [0.4.1] - 2021-01-07

2.4.1 Changed

- Move StateContext::Result to [State::Result](#) to remove circular include dependency

2.4.2 Removed

- Remove unused functions during code cleanup

2.5 [0.4.0] - 2020-12-03

2.5.1 Changed

- Switch off IIR filter on [SensorBME280](#)
- Use intf_ptr to pass SPI context to bme280_spi_* functions to avoid static objects

2.6 [0.3.1] - 2020-11-30

2.6.1 Changed

- Optimize imports, reformat code and document code

2.6.2 Fixed

- Set time to 2020-01-01T00:00:00 if time is not set, otherwise LoRa communication fails

2.7 [0.3.0] - 2020-11-23

2.7.1 Added

- Configuration option to disable sending status messages

2.7.2 Changed

- Change thing-ID and format for status messages

2.8 [0.2.0] - 2020-11-19

2.8.1 Added

- Configuration of correction offset and factor for every measured property
- Log [SensorSPS30](#) serial numbers and UUIDs to measurement data CSV files
- [Battery](#) class with function to read battery voltage
- Send a status message with battery voltage, RSSI and SNR in fixed time intervals via LoRa

2.8.2 Changed

- Wait 2-10min instead of 10-60s before retry to authenticate at gateway
- Reboot again after one hour if an error occurred
- Print statistics on status button press if PRINT_STAT is defined

2.9 [0.1.0] - 2020-11-10

2.9.1 Added

- Functions to calibrate RTC
- [SpsCommand](#) and [BmeCommand](#) to control sensors via USB interface
- [InfoCommand](#) to display SWUUID, HWUUID and software version
- Possibility to restore default configuration by pressing both buttons during power on

2.9.2 Changed

- Store and send only latest measurement values instead of maintaining a queue

2.9.3 Fixed

- Fix LoRa module state switch delay bug

2.10 [0.0.1] - 2020-11-02

2.10.1 Added

- All source files with initial implementation

Chapter 3

Namespace Index

3.1 Namespace List

Here is a list of all namespaces with brief descriptions:

utils	23
-----------------------	-------	--------------------

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

MeasureState::avg_data_s	27
Battery	28
SensorBME280::bme280_handle_t	29
Command	35
BmeCommand	30
ConfigCommand	56
InfoCommand	61
SpsCommand	106
TimeCommand	113
CommandHandler	39
Config	50
Config::config_value_t	55
Config::entry_t	58
Logger	71
mbed::NonCopyable	
Buttons	32
Communication	46
LedIndicator	66
Lora	74
SDCard	93
SensorBME280	97
SensorSPS30	102
StateContext	111
Protocol	86
State	108
CommandState	43
ErrorState	59
InitState	62
MeasureState	79
version_number_s	116

Chapter 5

Class Index

5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

MeasureState::avg_data_s	27
Battery	28
SensorBME280::bme280_handle_t	29
BmeCommand	30
Buttons	32
Command	35
CommandHandler	39
CommandState	43
Communication	46
Config	50
Config::config_value_t	55
ConfigCommand	56
Config::entry_t	58
ErrorState	59
InfoCommand	61
InitState	62
LedIndicator	66
Logger	71
Lora	74
MeasureState	79
Protocol	86
SDCard	93
SensorBME280	97
SensorSPS30	102
SpsCommand	106
State	108
StateContext	111
TimeCommand	113
version_number_s	116

Chapter 6

File Index

6.1 File List

Here is a list of all files with brief descriptions:

src/main.cpp	139
src/command/BmeCommand.cpp	119
src/command/BmeCommand.h	119
src/command/Command.cpp	119
src/command/Command.h	120
src/command/CommandHandler.cpp	120
src/command/CommandHandler.h	120
src/command/ConfigCommand.cpp	120
src/command/ConfigCommand.h	121
src/command/InfoCommand.cpp	121
src/command/InfoCommand.h	121
src/command/SpsCommand.cpp	121
src/command/SpsCommand.h	121
src/command/TimeCommand.cpp	122
src/command/TimeCommand.h	122
src/communication/Communication.cpp	122
src/communication/Communication.h	123
src/communication/Protocol.cpp	123
src/communication/Protocol.h	124
src/config/Config.cpp	125
src/config/Config.h	125
src/defs/defines.h	125
src/driver/Battery.cpp	133
src/driver/Battery.h	134
src/driver/Buttons.cpp	134
src/driver/Buttons.h	134
src/driver/LedIndicator.cpp	135
src/driver/LedIndicator.h	135
src/driver/Lora.cpp	136
src/driver/Lora.h	136
src/driver/SDCard.cpp	137
src/driver/SDCard.h	137
src/driver/SensorBME280.cpp	137
src/driver/SensorBME280.h	137
src/driver/SensorSPS30.cpp	138

src/driver/SensorSPS30.h	138
src/logging/Logger.cpp	138
src/logging/Logger.h	138
src/state/CommandState.cpp	140
src/state/CommandState.h	140
src/state/ErrorMessage.cpp	140
src/state/ErrorMessage.h	141
src/state/InitState.cpp	141
src/state/InitState.h	141
src/state/MeasureState.cpp	142
src/state/MeasureState.h	142
src/state/State.cpp	142
src/state/State.h	142
src/state/StateContext.cpp	143
src/state/StateContext.h	143
src/utis/utis.cpp	143
src/utis/utis.h	144

Chapter 7

Namespace Documentation

7.1 utils Namespace Reference

Functions

- void [get_formatted_time_string](#) (const char *format_string, std::string &dest, time_t timestamp)
- bool [is_number](#) (const std::string &str)
- bool [string_to_bool](#) (const std::string &str, bool &dest)
- std::string [to_hex_string](#) (uint32_t number)
- std::string [app_version](#) ()
- uint16_t [calc_crc16](#) (const void *buf, int size)
- uint32_t [calc_crc32](#) (const void *buf, int size)
- uint32_t [get_rand](#) ()

7.1.1 Function Documentation

7.1.1.1 [app_version\(\)](#)

```
std::string utils::app_version ( )
```

Return the [version](#) as a string. (e.g. "1.3.2")

Returns

the version number string

7.1.1.2 [calc_crc16\(\)](#)

```
uint16_t utils::calc_crc16 (
    const void * buf,
    int size )
```

Calculate the 16-bit CRC checksum of `buf`.

Parameters

<i>buf</i>	the data buffer to calculate the CRC from
<i>size</i>	the size of the data buffer

Returns

the 16-bit CRC checksum

7.1.1.3 calc_crc32()

```
uint32_t utils::calc_crc32 (
    const void * buf,
    int size )
```

Calculate the 32-bit CRC checksum of *buf*.

Parameters

<i>buf</i>	the data buffer to calculate the CRC from
<i>size</i>	the size of the data buffer

Returns

the 32-bit CRC checksum

7.1.1.4 get_formatted_time_string()

```
void utils::get_formatted_time_string (
    const char * format_string,
    std::string & dest,
    time_t timestamp = 0 )
```

Get the current time in the format specified by *format_string* and write the result into *dest*.

Parameters

<i>format_string</i>	Format specifier for the <code>strftime()</code> function
<i>dest</i>	The string to write to
<i>timestamp</i>	Optional timestamp. If 0, current time is converted to string.

7.1.1.5 get_rand()

```
uint32_t utils::get_rand ( )
```

Get a random 32-bit number generated by the builtin TRNG.

Returns

a random 32-bit number

7.1.1.6 is_number()

```
bool utils::is_number (
    const std::string & str )
```

Check `str` represents a signed integer.

Parameters

<i>str</i>	The number string
------------	-------------------

Returns

true if `str` is an integer number, false otherwise

7.1.1.7 string_to_bool()

```
bool utils::string_to_bool (
    const std::string & str,
    bool & dest )
```

Check if `represents` a boolean value.

Parameters

<i>str</i>	The boolean string
<i>dest</i>	The destination to write the result to

Returns

true, if the `str` represents a boolean value, false if parsing failed

7.1.1.8 to_hex_string()

```
std::string utils::to_hex_string (  
    uint32_t number )
```

Convert a given `number` into a hex string.

Parameters

<i>number</i>	The number to convert
---------------	-----------------------

Returns

the string with the number in hex format

Chapter 8

Class Documentation

8.1 MeasureState::avg_data_s Struct Reference

Public Member Functions

- struct [avg_data_s](#) & [operator+=](#) (const [avg_data_s](#) &rhs)
- struct [avg_data_s](#) & [operator/=](#) (const size_t rhs)

Public Attributes

- float [temp](#)
- float [hum](#)
- float [press](#)
- float [pm2_5](#)
- float [particle_size](#)

8.1.1 Detailed Description

Struct used to store the measurement points for the moving average. Overloaded operators are used to calculate the average.

8.1.2 Member Function Documentation

8.1.2.1 operator+=()

```
struct avg\_data\_s& MeasureState::avg_data_s::operator+= (
    const avg\_data\_s & rhs ) [inline]
```

8.1.2.2 operator/=()

```
struct avg_data_s& MeasureState::avg_data_s::operator/= (
    const size_t rhs ) [inline]
```

8.1.3 Member Data Documentation

8.1.3.1 hum

```
float MeasureState::avg_data_s::hum
```

8.1.3.2 particle_size

```
float MeasureState::avg_data_s::particle_size
```

8.1.3.3 pm2_5

```
float MeasureState::avg_data_s::pm2_5
```

8.1.3.4 press

```
float MeasureState::avg_data_s::press
```

8.1.3.5 temp

```
float MeasureState::avg_data_s::temp
```

The documentation for this struct was generated from the following file:

- [src/state/MeasureState.h](#)

8.2 Battery Class Reference

```
#include <Battery.h>
```

Static Public Member Functions

- static float [read_voltage](#) ()

8.2.1 Detailed Description

Static class providing functionality to read battery status

8.2.2 Member Function Documentation

8.2.2.1 [read_voltage\(\)](#)

```
float Battery::read_voltage ( ) [static]
```

Read the battery level from the analog input.

Returns

the battery level in volts

The documentation for this class was generated from the following files:

- [src/driver/Battery.h](#)
- [src/driver/Battery.cpp](#)

8.3 SensorBME280::bme280_handle_t Struct Reference

Public Attributes

- mbed::DigitalOut * [bme280_cs](#)
- mbed::SPI * [spi](#)

8.3.1 Detailed Description

Structure for the handles which are passed to the [bme280_spi_*](#) functions to provide access to the [_spi](#) and the [_bme280_cs](#) object.

8.3.2 Member Data Documentation

8.3.2.1 bme280_cs

```
mbed::DigitalOut* SensorBME280::bme280_handle_t::bme280_cs
```

8.3.2.2 spi

```
mbed::SPI* SensorBME280::bme280_handle_t::spi
```

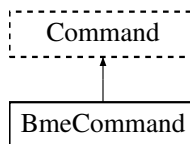
The documentation for this struct was generated from the following file:

- [src/driver/SensorBME280.h](#)

8.4 BmeCommand Class Reference

```
#include <BmeCommand.h>
```

Inheritance diagram for BmeCommand:



Public Member Functions

- [BmeCommand](#) (USBSerial &usb_serial, [SensorBME280](#) &bme280)
- [~BmeCommand](#) () override=default

Private Member Functions

- void [execute_command](#) (std::map< std::string, std::string > ¶meter_map, std::vector< std::string > ¶meter_vec) override

Private Attributes

- [SensorBME280](#) & [_bme280](#)

Additional Inherited Members

8.4.1 Detailed Description

Class handling commands to test and control the on-board BME280 sensor

8.4.2 Constructor & Destructor Documentation

8.4.2.1 BmeCommand()

```
BmeCommand::BmeCommand (
    USBSerial & usb_serial,
    SensorBME280 & bme280 )
```

Create a [BmeCommand](#) object

Parameters

<i>usb_serial</i>	reference to an USBSerial object
<i>bme280</i>	reference to the SensorBME280 object which should be controlled

8.4.2.2 ~BmeCommand()

```
BmeCommand::~~BmeCommand ( ) [override], [default]
```

Default destructor

8.4.3 Member Function Documentation

8.4.3.1 execute_command()

```
void BmeCommand::execute_command (
    std::map< std::string, std::string > & parameter_map,
    std::vector< std::string > & parameter_vec ) [override], [private], [virtual]
```

Executes the bme command according to the given parameters. The usage of the command is described in the [bme_command_description](#).

Parameters

<i>parameter_map</i>	map holding the parameters that are given as key-value-pairs
<i>parameter_vec</i>	vector holding the option parameters

Implements [Command](#).

8.4.4 Member Data Documentation

8.4.4.1 `_bme280`

`SensorBME280& BmeCommand::_bme280 [private]`

Reference to the [SensorBME280](#) object used to control the sensor by calling its member functions.

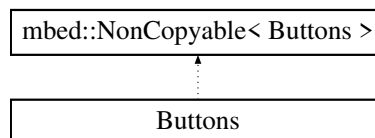
The documentation for this class was generated from the following files:

- [src/command/BmeCommand.h](#)
- [src/command/BmeCommand.cpp](#)

8.5 Buttons Class Reference

```
#include <Buttons.h>
```

Inheritance diagram for Buttons:



Public Member Functions

- [Buttons](#) (`rtos::EventFlags &event_flags`)
- `bool is_left_pressed ()`
- `bool is_right_pressed ()`

Private Member Functions

- `void button_left_pressed ()`
- `void button_right_pressed ()`
- `void button_left_released ()`
- `void button_right_released ()`

Private Attributes

- `rtos::EventFlags & _event_flags`
- `mbed::InterruptIn _button_left`
- `mbed::InterruptIn _button_right`
- `mbed::LowPowerTimer _button_left_timer`
- `mbed::LowPowerTimer _button_right_timer`

8.5.1 Detailed Description

Class handling button input

8.5.2 Constructor & Destructor Documentation

8.5.2.1 Buttons()

```
Buttons::Buttons (
    rtos::EventFlags & event_flags ) [explicit]
```

Create a [Buttons](#) object and attach the rise and fall callback functions to the [_button_left](#) and the [_button_right](#)

Parameters

<i>event_flags</i>	reference to an EventFlags object
--------------------	-----------------------------------

8.5.3 Member Function Documentation

8.5.3.1 button_left_pressed()

```
void Buttons::button_left_pressed ( ) [private]
```

Interrupt handler that is called at the falling edge of [_button_left](#).

8.5.3.2 button_left_released()

```
void Buttons::button_left_released ( ) [private]
```

Interrupt handler that is called at the rising edge of [_button_left](#). The [FLAG_BUTTON_MEASURE](#) is set in [_event_flags](#).

8.5.3.3 button_right_pressed()

```
void Buttons::button_right_pressed ( ) [private]
```

Interrupt handler that is called at the rising edge of [_button_right](#).

8.5.3.4 button_right_released()

```
void Buttons::button_right_released ( ) [private]
```

Interrupt handler that is called at the rising edge of [_button_right](#). Depending on the time how long the button was pressed, the [FLAG_BUTTON_STATUS](#) or the [FLAG_BUTTON_COMMAND](#) is set in [_event_flags](#).

8.5.3.5 is_left_pressed()

```
bool Buttons::is_left_pressed ( )
```

Function to check if [_button_left](#) is currently pressed

Returns

true if [_button_left](#) is pressed, false otherwise

8.5.3.6 is_right_pressed()

```
bool Buttons::is_right_pressed ( )
```

Function to check if [_button_right](#) is currently pressed

Returns

true if [_button_right](#) is pressed, false otherwise

8.5.4 Member Data Documentation

8.5.4.1 _button_left

```
mbed::InterruptIn Buttons::_button_left [private]
```

Interrupt input for the left button

8.5.4.2 _button_left_timer

```
mbed::LowPowerTimer Buttons::_button_left_timer [private]
```

Timer measuring how long the [_button_left](#) is pressed

8.5.4.3 `_button_right`

```
mbed::InterruptIn Buttons::_button_right [private]
```

Interrupt input for the right button

8.5.4.4 `_button_right_timer`

```
mbed::LowPowerTimer Buttons::_button_right_timer [private]
```

Timer measuring how long the `_button_right` is pressed

8.5.4.5 `_event_flags`

```
rtos::EventFlags& Buttons::_event_flags [private]
```

Reference to the EventFlags object, which is used to signal that a button was pressed.

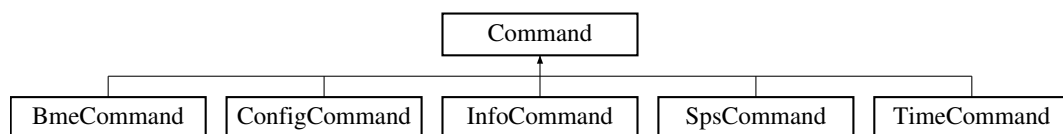
The documentation for this class was generated from the following files:

- [src/driver/Buttons.h](#)
- [src/driver/Buttons.cpp](#)

8.6 Command Class Reference

```
#include <Command.h>
```

Inheritance diagram for Command:



Public Member Functions

- [Command](#) (std::string name, std::string description, USBSerial &serial)
- virtual [~Command](#) ()=default
- void [execute](#) (const std::string ¶meter_string)
- const std::string & [get_name](#) () const
- const std::string & [get_description](#) () const

Protected Member Functions

- bool [extract_parameters](#) (const std::string &input_string, std::map< std::string, std::string > ¶meter_map, std::vector< std::string > ¶meter_vec)

Static Protected Member Functions

- static size_t [split_string](#) (const std::string &input_string, std::vector< std::string > &split_strings, char delimiter)

Protected Attributes

- USBSerial & [_usb_serial](#)

Private Member Functions

- virtual void [execute_command](#) (std::map< std::string, std::string > ¶meter_map, std::vector< std::string > ¶meter_vec)=0

Private Attributes

- const std::string [_name](#)
- const std::string [_description](#)

8.6.1 Detailed Description

Abstract [Command](#) base class

To add a new [Command](#), create a derived class which implements the method [execute_command\(\)](#).

8.6.2 Constructor & Destructor Documentation

8.6.2.1 Command()

```
Command::Command (
    std::string name,
    std::string description,
    USBSerial & serial )
```

Basic constructor

Parameters

<i>name</i>	Name of the command, should be defined in defines.h
<i>description</i>	Description and usage info for the command, should be defined in defines.h
<i>serial</i>	Reference to the USBSerial interface object

8.6.2.2 ~Command()

```
virtual Command::~~Command ( ) [virtual], [default]
```

Virtual destructor

8.6.3 Member Function Documentation

8.6.3.1 execute()

```
void Command::execute (
    const std::string & parameter_string )
```

Executes [extract_parameters\(\)](#) with the `parameter_string` and then the command specific [execute_command\(\)](#) is called with the extracted parameters.

Parameters

<code>parameter_string</code>	Command specific parameter string
-------------------------------	---------------------------------------------------

8.6.3.2 execute_command()

```
virtual void Command::execute_command (
    std::map< std::string, std::string > & parameter_map,
    std::vector< std::string > & parameter_vec ) [private], [pure virtual]
```

Implemented in [TimeCommand](#), [SpsCommand](#), [InfoCommand](#), [ConfigCommand](#), and [BmeCommand](#).

8.6.3.3 extract_parameters()

```
bool Command::extract_parameters (
    const std::string & input_string,
    std::map< std::string, std::string > & parameter_map,
    std::vector< std::string > & parameter_vec ) [protected]
```

Splits the `input_string` into parameters. Parameters are delimited by one or multiple spaces and can be single words, which are stored in the `parameter_vec`. Parameters can be also key-value-pairs (e.g. `key=value`), which are stored in the `parameter_map`.

Parameters

<code>input_string</code>	The string, including the parameters to be extracted.
<code>parameter_map</code>	Map where key-value-pair parameters are stored to.
<code>parameter_vec</code>	Vector where option parameters are stored to.

Returns

false, if `input_string` has an invalid format, true otherwise

8.6.3.4 get_description()

```
const std::string & Command::get_description ( ) const
```

Returns

reference to `_description`

8.6.3.5 get_name()

```
const std::string & Command::get_name ( ) const
```

Returns

reference to `_name`

8.6.3.6 split_string()

```
size_t Command::split_string (
    const std::string & input_string,
    std::vector< std::string > & split_strings,
    char delimiter ) [static], [protected]
```

Splits an `input_string` at a given delimiter and pushes the parts to the vector `split_strings`. Empty parts are not added to `split_strings`.

Parameters

<i>input_string</i>	String to be split
<i>split_strings</i>	Vector with the split parts of <code>input_string</code>
<i>delimiter</i>	Character on which the <code>input_string</code> is split

Returns

final size of `split_strings`

8.6.4 Member Data Documentation

8.6.4.1 `_description`

```
const std::string Command::_description [private]
```

Description and usage info for the command

8.6.4.2 `_name`

```
const std::string Command::_name [private]
```

[Command](#) name

8.6.4.3 `_usb_serial`

```
USBSerial& Command::_usb_serial [protected]
```

Reference to the USBSerial object used to read from and write to the USB virtual COM port

The documentation for this class was generated from the following files:

- `src/command/`[Command.h](#)
- `src/command/`[Command.cpp](#)

8.7 CommandHandler Class Reference

```
#include <CommandHandler.h>
```

Public Member Functions

- [CommandHandler](#) (USBSerial &usb_serial, mbed::FlashIAP &flash, [SensorSPS30](#) &sps30, [SensorBME280](#) &bme280, rtos::EventFlags &event_flags)
- [~CommandHandler](#) ()
- bool [connect](#) ()
- void [handle_usb_serial_command](#) ()
- void [disconnect](#) ()

Private Member Functions

- void [serial_interrupt_handler](#) ()
- void [print_help](#) ()
- void [print_prompt](#) ()
- void [read_line_from_buffer](#) (std::string &line)
- [Command](#) * [find_command](#) (const std::string &command_name)

Static Private Member Functions

- static void [separate_command_name](#) (const std::string &command_string, std::string &command_name, std::string ¶meters)

Private Attributes

- USBSerial & [_usb_serial](#)
- rtos::EventFlags & [_event_flags](#)
- std::vector< [Command](#) * > [_commands](#)
- mbed::CircularBuffer< char, [COMMAND_HANDLER_MAX_BUF_SIZE](#) > [_buffer](#)

8.7.1 Detailed Description

Class receiving and handling the commands from the USB virtual COM port interface

8.7.2 Constructor & Destructor Documentation

8.7.2.1 CommandHandler()

```
CommandHandler::CommandHandler (
    USBSerial & usb_serial,
    mbed::FlashIAP & flash,
    SensorSPS30 & sps30,
    SensorBME280 & bme280,
    rtos::EventFlags & event_flags )
```

Create a new [CommandHandler](#) and instantiate every command on the heap.

Parameters

<i>usb_serial</i>	reference to an USBSerial object
<i>flash</i>	reference to a FlashIAP object
<i>sps30</i>	reference to a SensorSPS30 object
<i>bme280</i>	reference to a SensorBME280 object
<i>event_flags</i>	reference to an EventFlags object

8.7.2.2 ~CommandHandler()

```
CommandHandler::~~CommandHandler ( )
```

The [CommandHandler](#) destructor deletes all [Command](#) objects from [_commands](#), which are stored on the heap

8.7.3 Member Function Documentation

8.7.3.1 connect()

```
bool CommandHandler::connect ( )
```

Tries to connect (non-blocking) to an USB serial terminal. If connection succeeds, [_buffer](#) is cleared and the software version and the command prompt are printed.

Returns

true if connected, false otherwise

8.7.3.2 disconnect()

```
void CommandHandler::disconnect ( )
```

Prints disconnect information and disconnects from the USB serial terminal and calls `USBSerial::deinit()` to unlock deepsleep again.

8.7.3.3 find_command()

```
Command * CommandHandler::find_command (
    const std::string & command_name ) [private]
```

Searches for a command in [_commands](#) with the given `command_name`.

Parameters

<i>command_name</i>	the name of the searched command
---------------------	----------------------------------

Returns

pointer to the [Command](#) object if the command exists, nullptr otherwise

8.7.3.4 handle_usb_serial_command()

```
void CommandHandler::handle_usb_serial_command ( )
```

Reads a line from [_buffer](#) and executes the given command with the given parameters if the command exists, otherwise a usage information with the supported commands is printed.

8.7.3.5 print_help()

```
void CommandHandler::print_help ( ) [private]
```

Print help message containing a list of all supported commands.

8.7.3.6 print_prompt()

```
void CommandHandler::print_prompt ( ) [private]
```

Prints the command prompt.

8.7.3.7 read_line_from_buffer()

```
void CommandHandler::read_line_from_buffer (
    std::string & line ) [private]
```

Pops one line from `_buffer` and pushes it to `line`.

Parameters

<i>line</i>	Reference to the string on which the line is pushed
-------------	-----------------------------------------------------

8.7.3.8 separate_command_name()

```
void CommandHandler::separate_command_name (
    const std::string & command_string,
    std::string & command_name,
    std::string & parameters ) [static], [private]
```

Splits the given `command_string` at the first space into the `command_name` and the `parameters`.

Parameters

<i>command_string</i>	whole command string
<i>command_name</i>	name of the command preceding the first space in <code>command_string</code>
<i>parameters</i>	parameters succeeding the first space in <code>command_string</code>

8.7.3.9 serial_interrupt_handler()

```
void CommandHandler::serial_interrupt_handler ( ) [private]
```

Interrupt handler that receives commands from the USB serial interface and stores them in `_buffer`. If a newline is received, the `FLAG_RECEIVED_COMMAND` flag is set in `_event_flags`.

8.7.4 Member Data Documentation

8.7.4.1 `_buffer`

```
mbed::CircularBuffer<char, COMMAND_HANDLER_MAX_BUF_SIZE> CommandHandler::_buffer [private]
```

Buffer used by [serial_interrupt_handler\(\)](#) to store received command strings until they are handled by [handle_usb_serial_command\(\)](#).

8.7.4.2 `_commands`

```
std::vector<Command *> CommandHandler::_commands [private]
```

Vector containing pointers to an instance of every command

8.7.4.3 `_event_flags`

```
rtos::EventFlags& CommandHandler::_event_flags [private]
```

Reference to the EventFlags object used to signal the [CommandState::handle\(\)](#) that a new command was received.

8.7.4.4 `_usb_serial`

```
USBSerial& CommandHandler::_usb_serial [private]
```

Reference to the USBSerial object handling all the low level operations for the connection, the input and the output.

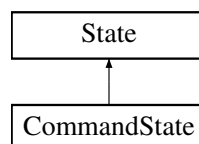
The documentation for this class was generated from the following files:

- [src/command/CommandHandler.h](#)
- [src/command/CommandHandler.cpp](#)

8.8 CommandState Class Reference

```
#include <CommandState.h>
```

Inheritance diagram for CommandState:



Public Member Functions

- [CommandState](#) ([LedIndicator](#) &led_indicator, [CommandHandler](#) &command_handler, [rtos::EventFlags](#) &event_flags)
- [~CommandState](#) () override=default
- [State::Result](#) [handle](#) () override

Private Member Functions

- [State::Result run \(\)](#)

Private Attributes

- [CommandHandler](#) & [_command_handler](#)
- [rtos::EventFlags](#) & [_event_flags](#)

Additional Inherited Members

8.8.1 Detailed Description

[State](#) that allows to execute commands received over the USB virtual COM port.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 CommandState()

```
CommandState::CommandState (
    LedIndicator & led_indicator,
    CommandHandler & command_handler,
    rtos::EventFlags & event_flags )
```

Create a [CommandState](#)

Parameters

<i>led_indicator</i>	Reference to the LedIndicator to set the LED behavior
<i>command_handler</i>	Reference to the CommandHandler doing the actual command handling
<i>event_flags</i>	Reference to the control event flags to react on button presses and incoming commands.

8.8.2.2 ~CommandState()

```
CommandState::~~CommandState ( ) [override], [default]
```

Default destructor

8.8.3 Member Function Documentation

8.8.3.1 handle()

```
State::Result CommandState::handle ( ) [override], [virtual]
```

Set the status LED color to orange and call [run\(\)](#) to handle the connection and incoming commands.

Returns

[State::Result::COMMAND_PRESSED](#)

Implements [State](#).

8.8.3.2 run()

```
State::Result CommandState::run ( ) [private]
```

Handle connection and incoming commands in the following order: Wait in an endless loop for a serial terminal to connect. If a serial terminal connected, start handling incoming commands by calling [CommandHandler::handle_usb_serial_command\(\)](#) if [FLAG_RECEIVED_COMMAND](#) is set.

Returns

[State::Result::COMMAND_PRESSED](#) if the [FLAG_BUTTON_COMMAND](#) was set

8.8.4 Member Data Documentation

8.8.4.1 _command_handler

```
CommandHandler& CommandState::_command_handler [private]
```

[CommandHandler](#) object used to connect and disconnect to the serial terminal and to dispatch the incoming commands.

8.8.4.2 _event_flags

```
rtos::EventFlags& CommandState::_event_flags [private]
```

Reference to the control event flags object to wait for control events to occur

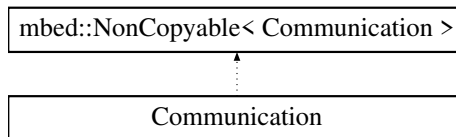
The documentation for this class was generated from the following files:

- [src/state/CommandState.h](#)
- [src/state/CommandState.cpp](#)

8.9 Communication Class Reference

```
#include <Communication.h>
```

Inheritance diagram for Communication:



Public Member Functions

- [Communication](#) ()
- [mbed_error_status_t start](#) ()
- void [stop](#) ()
- void [send_measured_data](#) (const [Protocol::measured_data_t](#) &data)

Private Types

- enum [PacketType](#) { [MEASURED_DATA](#) , [STATUS_DATA](#) }

Private Member Functions

- void [run](#) ()
- void [authenticate](#) ()
- bool [send_data](#) ([PacketType](#) packet_type)
- void [send_status_data](#) ()

Private Attributes

- [Lora_lora](#)
- [Protocol_protocol](#)
- [rtos::Thread_thread](#)
- [PlatformMutex_data_mutex](#)
- [Protocol::measured_data_t_measured_data](#) = {}
- [rtos::EventFlags_flags](#)
- [mbed::LowPowerTicker_status_update_ticker](#)

8.9.1 Detailed Description

Class handling the LoRa communication control flow. The LoRa communication is done in a separate thread, which can be started with [start\(\)](#) and stopped with [stop\(\)](#).

8.9.2 Member Enumeration Documentation

8.9.2.1 PacketType

```
enum Communication::PacketType [private]
```

Enum defining the possible data packet types

Enumerator

MEASURED_DATA	
STATUS_DATA	

8.9.3 Constructor & Destructor Documentation

8.9.3.1 Communication()

```
Communication::Communication ( )
```

Creates a [Communication](#) object

8.9.4 Member Function Documentation

8.9.4.1 authenticate()

```
void Communication::authenticate ( ) [private]
```

If sensor is not authenticated at the gateway, it will send an authentication request and wait for the gateway authentication response. If no or an invalid response is received, a new authentication message will be sent after a random delay between 2 and 10 minutes. This process will be repeated until a valid response is received.

8.9.4.2 run()

```
void Communication::run ( ) [private]
```

Endless running function executed in [_thread](#) handling the LoRa communication by reacting on event [_flags](#).

8.9.4.3 send_data()

```
bool Communication::send_data (
    PacketType packet_type ) [private]
```

Depending on the `packet_type`, a measurement data packet or a status update packet is sent to the gateway. If the gateway does not respond with an acknowledge, the packet will be sent again after a random delay between 2 and 10 minutes. If [MAX_SEND_DATA_ATTEMPTS](#) times no or an invalid response is received, the connection will be reset and a new authentication is necessary.

Parameters

<i>packet_type</i>	Type of the packet to send
--------------------	----------------------------

Returns

true if packet was sent, false otherwise

8.9.4.4 send_measured_data()

```
void Communication::send_measured_data (
    const Protocol::measured_data_t & data )
```

Copies `data` to `_measured_data` and sets the `FLAG_SEND_MEASURED_DATA` in `_flags`.

Parameters

<i>data</i>	measured data to be sent
-------------	--------------------------

8.9.4.5 send_status_data()

```
void Communication::send_status_data ( ) [private]
```

Initiates sending a status update packet by setting `FLAG_SEND_STATUS_DATA` in `_flags`.

8.9.4.6 start()

```
mbed_error_status_t Communication::start ( )
```

Initialize `_lora` and start the LoRa communication `_thread` with the function `run()`. If "status_enable" is true in `Config::global_config`, the `_status_update_ticker` is started to send periodic status updates with an interval defined by "status_interval_sec" in `Config::global_config`.

Returns

MBED_SUCCESS if thread was started, MBED_ERROR_FAILED_OPERATION otherwise

8.9.4.7 stop()

```
void Communication::stop ( )
```

If the LoRa communication `_thread` is running, it will be stopped, as well as the `_status_update_ticker`. The `_lora` module is set to sleep mode.

8.9.5 Member Data Documentation

8.9.5.1 `_data_mutex`

```
PlatformMutex Communication::_data_mutex [private]
```

Mutex to serialize read and write accesses to [_measured_data](#)

8.9.5.2 `_flags`

```
rtos::EventFlags Communication::_flags [private]
```

EventFlags to signalize [_thread](#), when to send measurement data or a status update

8.9.5.3 `_lora`

```
Lora Communication::_lora [private]
```

[Lora](#) object used to control the LoRa module

8.9.5.4 `_measured_data`

```
Protocol::measured_data_t Communication::_measured_data = {} [private]
```

Buffer for the latest measured data to be sent next

8.9.5.5 `_protocol`

```
Protocol Communication::_protocol [private]
```

[Protocol](#) object providing functions to communicate with the Smartmote gateway

8.9.5.6 `_status_update_ticker`

```
mbed::LowPowerTicker Communication::_status_update_ticker [private]
```

Ticker to periodically initiate sending a status update

8.9.5.7 _thread

```
rtos::Thread Communication::_thread [private]
```

The communication thread

The documentation for this class was generated from the following files:

- src/communication/[Communication.h](#)
- src/communication/[Communication.cpp](#)

8.10 Config Class Reference

```
#include <Config.h>
```

Classes

- union [config_value_t](#)
- struct [entry_t](#)

Public Types

- enum [ConfigType](#) { [BOOL](#) , [UINT32](#) , [FLOAT](#) }

Static Public Member Functions

- static void [save_default_config](#) ()
- static void [reset_config](#) ()
- template<typename T >
static T [get](#) (const std::string &name)
- static bool [get_entry](#) (const std::string &name, [entry_t](#) &dest)
- static bool [set_value](#) (const std::string &name, const std::string &value_string)
- static mbed_error_status_t [write_config_to_flash](#) (mbed::FlashIAP &flash)
- static mbed_error_status_t [read_config_from_flash](#) (mbed::FlashIAP &flash)

Static Public Attributes

- static [entry_t](#) [global_config](#) [[NUM_CONFIG_ENTRIES](#)]

Static Private Member Functions

- static mbed_error_status_t [write_to_flash](#) (mbed::FlashIAP &flash, uint32_t address, uint8_t *source, size_t size)
- static mbed_error_status_t [read_from_flash](#) (mbed::FlashIAP &flash, uint32_t address, uint8_t *destination, size_t size)

Static Private Attributes

- static `Config::config_value_t default_config [NUM_CONFIG_ENTRIES] = {0}`

8.10.1 Detailed Description

Class holding only static member functions and static variables to provide an easy access to the configuration parameters

8.10.2 Member Enumeration Documentation

8.10.2.1 ConfigType

enum `Config::ConfigType`

Enum defining the possible data types for the config entries

Enumerator

BOOL	
UINT32	
FLOAT	

8.10.3 Member Function Documentation

8.10.3.1 get()

```
template<typename T >
T Config::get (
    const std::string & name ) [static]
```

Getter function for configuration parameter values.

Template Parameters

<i>T</i>	Data type of the parameter to be retrieved
----------	--------------------------------------------

Parameters

<i>name</i>	Name of the parameter to be retrieved
-------------	---------------------------------------

Returns

the value of the parameter with the given `name`

8.10.3.2 get_entry()

```
bool Config::get_entry (
    const std::string & name,
    entry_t & dest ) [static]
```

Getter function for a configuration parameter entry

Parameters

<i>name</i>	Name of the parameter to be retrieved
<i>dest</i>	Destination on which the entry is written to

Returns

true if a parameter with the given `name` was found, false otherwise

8.10.3.3 read_config_from_flash()

```
mbd_error_status_t Config::read_config_from_flash (
    mbed::FlashIAP & flash ) [static]
```

Reads all configuration parameter values from the flash memory at address [STORAGE_ADDR_CONFIG](#).

Parameters

<i>flash</i>	
--------------	--

Returns**8.10.3.4 read_from_flash()**

```
mbd_error_status_t Config::read_from_flash (
    mbed::FlashIAP & flash,
    uint32_t address,
    uint8_t * destination,
    size_t size ) [static], [private]
```

Parameters

<i>flash</i>	Reference to the FlashIAP object
<i>address</i>	Address within the flash memory to read from
<i>destination</i>	Buffer where the read data is copied to
<i>size</i>	Size of the data to be read

Returns

MBED_SUCCESS if read was successful, an `mbed_error_status_t` otherwise

8.10.3.5 reset_config()

```
void Config::reset_config ( ) [static]
```

Copies the the values from [default_config](#) to [global_config](#).

8.10.3.6 save_default_config()

```
void Config::save_default_config ( ) [static]
```

Copies the parameter values of [global_config](#) to [default_config](#). This function is used to store the default configuration before the [global_config](#) is overwritten with the values loaded from the flash memory.

8.10.3.7 set_value()

```
bool Config::set_value (
    const std::string & name,
    const std::string & value_string ) [static]
```

Set the parameter with the given `name` to the value of the given `value_string` if the `name` exists and the value is within the range of the parameter.

Parameters

<i>name</i>	Name of the parameter to be set
<i>value_string</i>	The value to be set as string

Returns

true if the parameter was set, false otherwise

8.10.3.8 write_config_to_flash()

```
mbed_error_status_t Config::write_config_to_flash (
    mbed::FlashIAP & flash ) [static]
```

Writes all configuration parameter values to the flash memory at address [STORAGE_ADDR_CONFIG](#).

Parameters

<i>flash</i>	Reference to the FlashIAP object
--------------	----------------------------------

Returns

MBED_SUCCESS if write operation succeeded, an mbed_error_status_t otherwise

8.10.3.9 write_to_flash()

```
mbed_error_status_t Config::write_to_flash (
    mbed::FlashIAP & flash,
    uint32_t address,
    uint8_t * source,
    size_t size ) [static], [private]
```

Writes data from *source* with the given *size* to the given *address* in the flash memory.

Parameters

<i>flash</i>	Reference to the FlashIAP object
<i>address</i>	Address within the flash memory to write to
<i>source</i>	Data which is written to the flash memory
<i>size</i>	Size of the data to be written

Returns

MBED_SUCCESS if write was successful, an mbed_error_status_t otherwise

8.10.4 Member Data Documentation

8.10.4.1 default_config

```
Config::config\_value\_t Config::default_config = {0} [static], [private]
```

Array holding the default configuration parameter values.

8.10.4.2 global_config

```
Config::entry_t Config::global_config [static]
```

Array holding the configuration parameters including the meta data for every parameter.

The documentation for this class was generated from the following files:

- [src/config/Config.h](#)
- [src/config/Config.cpp](#)

8.11 Config::config_value_t Union Reference

```
#include <Config.h>
```

Public Attributes

- bool [bool_v](#)
- uint32_t [uint32_v](#)
- float [float_v](#)

8.11.1 Detailed Description

Union defining the config value types

8.11.2 Member Data Documentation

8.11.2.1 bool_v

```
bool Config::config_value_t::bool_v
```

8.11.2.2 float_v

```
float Config::config_value_t::float_v
```

8.11.2.3 uint32_v

```
uint32_t Config::config_value_t::uint32_v
```

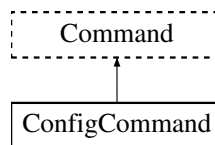
The documentation for this union was generated from the following file:

- [src/config/Config.h](#)

8.12 ConfigCommand Class Reference

```
#include <ConfigCommand.h>
```

Inheritance diagram for ConfigCommand:



Public Member Functions

- [ConfigCommand](#) (USBSerial &usb_serial, mbed::FlashIAP &flash)
- [~ConfigCommand](#) () override=default

Private Member Functions

- void [execute_command](#) (std::map< std::string, std::string > ¶meter_map, std::vector< std::string > ¶meter_vec) override
- void [print_current_config](#) ()
- void [handle_parameters](#) (const std::map< std::string, std::string > ¶meter_map)

Private Attributes

- mbed::FlashIAP & [_flash](#)

Additional Inherited Members

8.12.1 Detailed Description

Class handling commands to change various configuration parameters

8.12.2 Constructor & Destructor Documentation

8.12.2.1 ConfigCommand()

```
ConfigCommand::ConfigCommand (
    USBSerial & usb_serial,
    mbed::FlashIAP & flash )
```

Create a [ConfigCommand](#) object

Parameters

<i>usb_serial</i>	reference to an USBSerial object
<i>flash</i>	reference to a FlashIAP object

8.12.2.2 ~ConfigCommand()

```
ConfigCommand::~ConfigCommand ( ) [override], [default]
```

Default destructor

8.12.3 Member Function Documentation**8.12.3.1 execute_command()**

```
void ConfigCommand::execute_command (
    std::map< std::string, std::string > & parameter_map,
    std::vector< std::string > & parameter_vec ) [override], [private], [virtual]
```

Executes the config command according to the given parameters. The usage of the command is described in the [config_command_description](#).

Parameters

<i>parameter_map</i>	map holding the parameters that are given as key-value-pairs
<i>parameter_vec</i>	vector holding the option parameters

Implements [Command](#).

8.12.3.2 handle_parameters()

```
void ConfigCommand::handle_parameters (
    const std::map< std::string, std::string > & parameter_map ) [private]
```

Sets the configuration parameters in [Config::global_config](#) according to the given parameters. The configuration is then written to the flash memory at address [STORAGE_ADDR_CONFIG](#) to store the configuration persistently.

Parameters

<i>parameter_map</i>	Map with parameters as key-value pairs
----------------------	----------------------------------------

8.12.3.3 print_current_config()

```
void ConfigCommand::print_current_config ( ) [private]
```

Prints the current configuration

8.12.4 Member Data Documentation

8.12.4.1 _flash

```
mbed::FlashIAP& ConfigCommand::_flash [private]
```

Referece to the FlashIAP object which handles the low level operations to store the configuration in the flash memory.

The documentation for this class was generated from the following files:

- src/command/[ConfigCommand.h](#)
- src/command/[ConfigCommand.cpp](#)

8.13 Config::entry_t Struct Reference

```
#include <Config.h>
```

Public Attributes

- std::string [name](#)
- [ConfigType](#) [type](#)
- [config_value_t](#) [value](#)
- [config_value_t](#) [min](#)
- [config_value_t](#) [max](#)
- std::string [description](#)

8.13.1 Detailed Description

Defines the format of a config parameter entry

8.13.2 Member Data Documentation

8.13.2.1 description

`std::string Config::entry_t::description`

8.13.2.2 max

`config_value_t Config::entry_t::max`

8.13.2.3 min

`config_value_t Config::entry_t::min`

8.13.2.4 name

`std::string Config::entry_t::name`

8.13.2.5 type

`ConfigType Config::entry_t::type`

8.13.2.6 value

`config_value_t Config::entry_t::value`

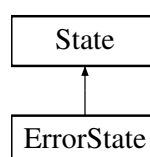
The documentation for this struct was generated from the following file:

- [src/config/Config.h](#)

8.14 ErrorState Class Reference

```
#include <ErrorState.h>
```

Inheritance diagram for ErrorState:



Public Member Functions

- [ErrorState](#) ([LedIndicator](#) &led_indicator, rtos::EventFlags &control_event_flags)
- [State::Result](#) handle () override

Private Attributes

- rtos::EventFlags & [_control_event_flags](#)

Additional Inherited Members

8.14.1 Detailed Description

Error state that is entered if any other state returned with an error.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 ErrorState()

```
ErrorState::ErrorState (
    LedIndicator & led_indicator,
    rtos::EventFlags & control_event_flags )
```

Create an [ErrorState](#)

Parameters

<i>led_indicator</i>	Reference to the LedIndicator to control the status LED
<i>control_event_flags</i>	Reference to the control event flags to react on button presses

8.14.3 Member Function Documentation

8.14.3.1 handle()

```
State::Result ErrorState::handle ( ) [override], [virtual]
```

Set the status LED to red for 3 seconds and start reboot. If an error occurs for MBED_CONF_PLATFORM_↔ ERROR_REBOOT_MAX times, the system will wait for one hour and then reboot again.

Returns

Function never returns, only calls MBED_ERROR(), which results in a reboot

Implements [State](#).

8.14.4 Member Data Documentation

8.14.4.1 `_control_event_flags`

```
rtos::EventFlags& ErrorState::_control_event_flags [private]
```

Reference to the control event flags object to wait for control events to occur

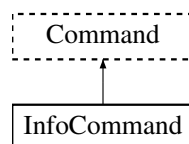
The documentation for this class was generated from the following files:

- [src/state/ErrorState.h](#)
- [src/state/ErrorState.cpp](#)

8.15 InfoCommand Class Reference

```
#include <InfoCommand.h>
```

Inheritance diagram for InfoCommand:



Public Member Functions

- [InfoCommand](#) (USBSerial &usb_serial)
- [~InfoCommand](#) () override=default

Private Member Functions

- void [execute_command](#) (std::map< std::string, std::string > ¶meter_map, std::vector< std::string > ¶meter_vec) override

Additional Inherited Members

8.15.1 Detailed Description

Class handling commands to print version and UUID information about the sensor system.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 InfoCommand()

```
InfoCommand::InfoCommand (
    USBSerial & usb_serial ) [explicit]
```

Create a [InfoCommand](#) object

Parameters

<i>usb_serial</i>	reference to an USBSerial object
-------------------	----------------------------------

8.15.2.2 ~InfoCommand()

```
InfoCommand::~~InfoCommand ( ) [override], [default]
```

Default destructor

8.15.3 Member Function Documentation**8.15.3.1 execute_command()**

```
void InfoCommand::execute_command (
    std::map< std::string, std::string > & parameter_map,
    std::vector< std::string > & parameter_vec ) [override], [private], [virtual]
```

Executes the info command and prints the versions and UUIDs. The usage of the command is described in the [info_command_description](#).

Parameters

<i>parameter_map</i>	map holding the parameters that are given as key-value-pairs
<i>parameter_vec</i>	vector holding the option parameters

Implements [Command](#).

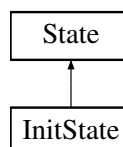
The documentation for this class was generated from the following files:

- [src/command/InfoCommand.h](#)
- [src/command/InfoCommand.cpp](#)

8.16 InitState Class Reference

```
#include <InitState.h>
```

Inheritance diagram for InitState:



Public Member Functions

- [InitState](#) ([LedIndicator](#) &led_indicator, [SDCard](#) &sd_card, [Communication](#) &communication, mbed::FlashIAP &flash, [SensorSPS30](#) &sps30, [SensorBME280](#) &bme280, [Buttons](#) &buttons)
- [~InitState](#) () override=default
- [State::Result](#) [handle](#) () override

Private Member Functions

- [State::Result](#) [init](#) ()
- mbed_error_status_t [sps30_test](#) ()
- mbed_error_status_t [bme280_test](#) ()
- void [init_rtc](#) ()

Private Attributes

- [SDCard](#) & [_sd_card](#)
- [Communication](#) & [_communication](#)
- mbed::FlashIAP & [_flash](#)
- [SensorSPS30](#) & [_sps30](#)
- [SensorBME280](#) & [_bme280](#)
- [Buttons](#) & [_buttons](#)

Additional Inherited Members

8.16.1 Detailed Description

[State](#) initializing all objects which need explicit initialization. Also test if connected sensors are working.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 InitState()

```
InitState::InitState (  
    LedIndicator & led_indicator,  
    SDCard & sd_card,  
    Communication & communication,  
    mbed::FlashIAP & flash,  
    SensorSPS30 & sps30,  
    SensorBME280 & bme280,  
    Buttons & buttons )
```

Create an Initstate

Parameters

<i>led_indicator</i>	Reference to the LedIndicator to control the status LED
<i>sd_card</i>	Reference to the SDCard to initialize it and also the Logger
<i>communication</i>	Reference to the Communication object to start the LoRa communication thread
<i>flash</i>	Reference to the FlashIAP to load config from flash
<i>sps30</i>	Reference to the SensorSPS30 to perform a start up test
<i>bme280</i>	Reference to the SensorBME280 to perform a start up test
<i>buttons</i>	Reference to the Buttons , to check if default config should be loaded

8.16.2.2 ~InitState()

```
InitState::~~InitState ( ) [override], [default]
```

Default destructor

8.16.3 Member Function Documentation**8.16.3.1 bme280_test()**

```
mbed_error_status_t InitState::bme280_test ( ) [private]
```

Initialize the [SensorBME280](#) and test if the BME280 is working by reading measurement values and checking if they are in range.

Returns

MBED_SUCCESS if SPS30 works, an mbed_error_status_t otherwise

8.16.3.2 handle()

```
State::Result InitState::handle ( ) [override], [virtual]
```

Set status LED to orange, execute [init\(\)](#) and depending on the return value, set status LED to green or red for 3 seconds.

Returns

the return value of [init\(\)](#)

Implements [State](#).

8.16.3.3 init()

```
State::Result InitState::init ( ) [private]
```

Initialize the [_sd_card](#), RTC, [Logger](#), [SensorSPS30](#) and [SensorBME280](#) and the [_flash](#). If the two buttons are pressed, the [Config::global_config](#) is set to its default values. Otherwise, the stored config is loaded from flash memory and written to [Config::global_config](#).

Returns

[State::Result::SUCCESS](#) on success, an [State::Result](#)

8.16.3.4 init_rtc()

```
void InitState::init_rtc ( ) [private]
```

If the RTC time is not set, i.e. it is set to a date earlier than [TIMESTAMP_20200101](#), it will be set to [TIMESTAMP_20200101](#). Note: This is necessary, otherwise odd timing problems occur.

8.16.3.5 sps30_test()

```
mbed_error_status_t InitState::sps30_test ( ) [private]
```

Test if the SPS30 is working by executing the [SensorSPS30::probe\(\)](#) function.

Returns

MBED_SUCCESS if SPS30 works, an [mbed_error_status_t](#) otherwise

8.16.4 Member Data Documentation

8.16.4.1 _bme280

```
SensorBME280& InitState::_bme280 [private]
```

Reference to [SensorBME280](#) to test if BME280 is working

8.16.4.2 _buttons

```
Buttons& InitState::_buttons [private]
```

Reference to [Buttons](#). If both buttons are pressed, the default configuration is used for [Config::default_config](#), otherwise the configuration is loaded from [_flash](#).

8.16.4.3 `_communication`

```
Communication& InitState::_communication [private]
```

Reference to the [Communication](#) object to start the LoRa communication thread.

8.16.4.4 `_flash`

```
mbed::FlashIAP& InitState::_flash [private]
```

Reference to the FlashIAP to initialize it and load stored configurations to [Config::global_config](#).

8.16.4.5 `_sd_card`

```
SDCard& InitState::_sd_card [private]
```

Reference to the [SDCard](#) to initialize it and the [Logger](#)

8.16.4.6 `_sps30`

```
SensorSPS30& InitState::_sps30 [private]
```

Reference to [SensorSPS30](#) to test if SPS30 is working

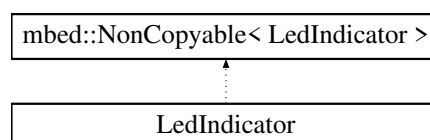
The documentation for this class was generated from the following files:

- [src/state/InitState.h](#)
- [src/state/InitState.cpp](#)

8.17 LedIndicator Class Reference

```
#include <LedIndicator.h>
```

Inheritance diagram for LedIndicator:



Public Types

- enum class [Color](#) { [OFF](#) , [GREEN](#) , [RED](#) , [ORANGE](#) }

Public Member Functions

- [LedIndicator](#) ()
- [~LedIndicator](#) ()=default
- void [display_battery_level_for](#) (uint32_t duration_ms)
- void [display_status_for](#) (Color color, uint32_t duration_ms=0)

Private Member Functions

- void [switch_battery_leds](#) (float level)
- void [turn_off_battery_leds](#) ()
- void [switch_status_leds](#) (Color color)
- void [turn_off_status_leds](#) ()

Private Attributes

- mbed::DigitalOut [_bat_led_1](#)
- mbed::DigitalOut [_bat_led_2](#)
- mbed::DigitalOut [_bat_led_3](#)
- mbed::DigitalOut [_status_led_green](#)
- mbed::DigitalOut [_status_led_red](#)
- mbed::LowPowerTimeout [_bat_led_timeout](#)
- mbed::LowPowerTimeout [_status_led_timeout](#)

8.17.1 Detailed Description

Class handling the behaviour of the battery LEDs and the status LED.

8.17.2 Member Enumeration Documentation

8.17.2.1 Color

```
enum LedIndicator::Color [strong]
```

Enum defining the possible states for the bicolor status LED.

Enumerator

OFF	
GREEN	
RED	
ORANGE	

8.17.3 Constructor & Destructor Documentation

8.17.3.1 LedIndicator()

```
LedIndicator::LedIndicator ( )
```

Creates a new [LedIndicator](#) object

8.17.3.2 ~LedIndicator()

```
LedIndicator::~~LedIndicator ( ) [default]
```

Default destructor

8.17.4 Member Function Documentation

8.17.4.1 display_battery_level_for()

```
void LedIndicator::display_battery_level_for (
    uint32_t duration_ms )
```

Displays the battery level on the battery LEDs for the given time.

Parameters

<i>duration_ms</i>	Time in milliseconds for how long the battery level is displayed
--------------------	------------------------------------------------------------------

8.17.4.2 display_status_for()

```
void LedIndicator::display_status_for (
    Color color,
    uint32_t duration_ms = 0 )
```

Displays the given `color` on the status LEDs for the given time.

Parameters

<i>color</i>	The color to be displayed
<i>duration_ms</i>	Time in milliseconds for how long the status is displayed

8.17.4.3 switch_battery_leds()

```
void LedIndicator::switch_battery_leds (
    float level ) [private]
```

Switch the battery LEDs according to the given battery level using the defined battery thresholds from [LedIndicator.cpp](#).

Parameters

<i>level</i>	The battery level in volts to be displayed
--------------	--------------------------------------------

8.17.4.4 switch_status_leds()

```
void LedIndicator::switch_status_leds (
    Color color ) [private]
```

Switch the status LED to the given color

Parameters

<i>color</i>	The color to be displayed
--------------	---------------------------

8.17.4.5 turn_off_battery_leds()

```
void LedIndicator::turn_off_battery_leds ( ) [private]
```

Turn off all battery LEDs.

8.17.4.6 turn_off_status_leds()

```
void LedIndicator::turn_off_status_leds ( ) [private]
```

Turn off the status LED

8.17.5 Member Data Documentation

8.17.5.1 `_bat_led_1`

```
mbed::DigitalOut LedIndicator::_bat_led_1 [private]
```

The digital output for the left most battery LED

8.17.5.2 `_bat_led_2`

```
mbed::DigitalOut LedIndicator::_bat_led_2 [private]
```

The digital output for the middle battery LED

8.17.5.3 `_bat_led_3`

```
mbed::DigitalOut LedIndicator::_bat_led_3 [private]
```

The digital output for the right most battery LED

8.17.5.4 `_bat_led_timeout`

```
mbed::LowPowerTimeout LedIndicator::_bat_led_timeout [private]
```

Timeout for the battery LEDs used to switch them off after a specified timeout.

8.17.5.5 `_status_led_green`

```
mbed::DigitalOut LedIndicator::_status_led_green [private]
```

The digital output for the green part of the status LED

8.17.5.6 `_status_led_red`

```
mbed::DigitalOut LedIndicator::_status_led_red [private]
```

The digital output for the red part of the status LED

8.17.5.7 `_status_led_timeout`

```
mbed::LowPowerTimeout LedIndicator::_status_led_timeout [private]
```

Timeout for the status LED used to switch it off after a specified timeout.

The documentation for this class was generated from the following files:

- [src/driver/LedIndicator.h](#)
- [src/driver/LedIndicator.cpp](#)

8.18 Logger Class Reference

```
#include <Logger.h>
```

Static Public Member Functions

- static void [init](#) ([SDCard](#) *sd_card)
- static void [log_internal](#) (const char *file_name, [LogLevel](#) log_level, const char *format_string,...)

Private Member Functions

- [Logger](#) ()=default
- [~Logger](#) ()=default
- [Logger](#) (const [Logger](#) &)
- [Logger](#) & [operator=](#) (const [Logger](#) &)

Static Private Member Functions

- static void [compose_log_line](#) (std::string &log_line, const char *file_name, [LogLevel](#) log_level, const std::string &message)

Static Private Attributes

- static bool [_initialized](#) = false
- static [SDCard](#) * [_sd_card](#) = nullptr
- static std::string [_file_name](#) = "log.txt"

8.18.1 Detailed Description

Class providing logging functionality. This class does not need to and can not be instantiated, because all members and member functions are static and the constructor is private. Instead it is necessary to initialize the [Logger](#) once with [init\(\)](#)

8.18.2 Constructor & Destructor Documentation

8.18.2.1 [Logger\(\)](#) [1/2]

```
Logger::Logger ( ) [private], [default]
```

Private constructor

8.18.2.2 ~Logger()

```
Logger::~Logger ( ) [private], [default]
```

Private destructor

8.18.2.3 Logger() [2/2]

```
Logger::Logger (
    const Logger & ) [private]
```

Private copy constructor

8.18.3 Member Function Documentation

8.18.3.1 compose_log_line()

```
void Logger::compose_log_line (
    std::string & log_line,
    const char * file_name,
    LogLevel log_level,
    const std::string & message ) [static], [private]
```

Composes a log entry out of the *file_name*, the *log_level* name and the log message and writes it to *log_line*.

Parameters

<i>log_line</i>	The destination where the composed log line is written to
<i>file_name</i>	The file name where the LOG was called from
<i>log_level</i>	The log level
<i>message</i>	The message to be logged

8.18.3.2 init()

```
void Logger::init (
    SDCard * sd_card ) [static]
```

Initialize the [Logger](#). Create a new log file on the SD card.

Parameters

<i>sd_card</i>	Pointer to the SD card object
----------------	-------------------------------

8.18.3.3 log_internal()

```
void Logger::log_internal (
    const char * file_name,
    LogLevel log_level,
    const char * format_string,
    ... ) [static]
```

Add a log line to the log file if given `log_level` is lower than the one in the [Config::global_config](#).

Parameters

<i>file_name</i>	File name where the function is called from
<i>log_level</i>	Log level
<i>format_string</i>	Format string for <code>sprintf()</code>
...	Argument list for <code>sprintf()</code>

8.18.3.4 operator=()

```
Logger& Logger::operator= (
    const Logger & ) [private]
```

Private assignment operator

8.18.4 Member Data Documentation

8.18.4.1 _file_name

```
std::string Logger::_file_name = "log.txt" [static], [private]
```

File name of the currently used log file

8.18.4.2 _initialized

```
bool Logger::_initialized = false [static], [private]
```

Flag to keep track if the [Logger](#) is already initialized or not. Is set to true in the [init\(\)](#) function

8.18.4.3 `_sd_card`

```
SDCard * Logger::_sd_card = nullptr [static], [private]
```

Pointer to the SD card object

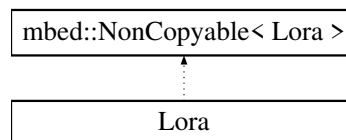
The documentation for this class was generated from the following files:

- src/logging/[Logger.h](#)
- src/logging/[Logger.cpp](#)

8.19 Lora Class Reference

```
#include <Lora.h>
```

Inheritance diagram for Lora:



Public Member Functions

- [Lora](#) ()
- void [init](#) ()
- void [sleep](#) ()
- void [send](#) (uint8_t *data, uint8_t size)
- mbed_error_status_t [receive](#) ()
- void [set_tx_config](#) ()
- void [set_rx_config](#) ()

Public Attributes

- uint8_t [rx_payload](#) [MAX_DATA_BUFFER_SIZE_SX126X]
- uint16_t [rx_payload_len](#) = 0
- int8_t [rx_rssi](#) = 0
- int8_t [rx_snr](#) = 0

Private Member Functions

- void [tx_done_cb](#) ()
- void [rx_error_cb](#) ()
- void [tx_timeout_cb](#) ()
- void [rx_timeout_cb](#) ()
- void [cad_done_cb](#) (bool arg)
- void [rx_done_cb](#) (const uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr)
- void [fhss_change_channel_cb](#) (uint8_t channel)

Private Attributes

- [SX126X_LoRaRadio _radio](#)
- [radio_events_t _radio_callbacks](#)
- [rtos::EventFlags _event_flags](#)
- [mbed::LowPowerTimeout _rx_timeout](#)

8.19.1 Detailed Description

Class acting as an adapter to provide simple functions to control the LoRa module.

8.19.2 Constructor & Destructor Documentation

8.19.2.1 Lora()

```
Lora::Lora ( )
```

Create a new [Lora](#) object

8.19.3 Member Function Documentation

8.19.3.1 cad_done_cb()

```
void Lora::cad_done_cb (
    bool arg ) [private]
```

Callback function which is called as soon as the channel activity detection is done.(not used)

Parameters

<i>arg</i>	true, if channel activity detected
------------	------------------------------------

8.19.3.2 fhss_change_channel_cb()

```
void Lora::fhss_change_channel_cb (
    uint8_t channel ) [private]
```

FHSS Change Channel callback (not used)

Parameters

<i>channel</i>	The index number of the current channel
----------------	-----------------------------------------

8.19.3.3 init()

```
void Lora::init ( )
```

Initialize [Lora](#) object by setting the [_radio_callbacks](#) initializing the [_radio](#) and setting radio specific parameters.

8.19.3.4 receive()

```
mbed_error_status_t Lora::receive ( )
```

Wait to receive a LoRa packet for a maximum time of [LORA_RX_TIMEOUT_MS](#).

Returns

MBED_SUCCESS if a packet was received, MBED_ERROR_TIME_OUT on timeout

8.19.3.5 rx_done_cb()

```
void Lora::rx_done_cb (
    const uint8_t * payload,
    uint16_t size,
    int16_t rssi,
    int8_t snr ) [private]
```

Callback function which is called if a LoRa packet was received.

Parameters

<i>payload</i>	The received payload
<i>size</i>	Size of the received payload
<i>rssi</i>	RSSI of the received packet
<i>snr</i>	SNR of the received packet

8.19.3.6 rx_error_cb()

```
void Lora::rx_error_cb ( ) [private]
```

Callback function which is called on a receive error

8.19.3.7 rx_timeout_cb()

```
void Lora::rx_timeout_cb ( ) [private]
```

Callback function which is called if a receive timeout occurs

8.19.3.8 send()

```
void Lora::send (
    uint8_t * data,
    uint8_t size )
```

Set LoRa module into standby mode, send `data` of the given `size` via the LoRa module and wait until the transmission is done.

Parameters

<i>data</i>	Data to be sent
<i>size</i>	Size of the data

8.19.3.9 set_rx_config()

```
void Lora::set_rx_config ( )
```

Set the RX parameters of [_radio](#) to the values defined in the [Config::global_config](#)

8.19.3.10 set_tx_config()

```
void Lora::set_tx_config ( )
```

Set the TX parameters of [_radio](#) to the values defined in the [Config::global_config](#)

8.19.3.11 sleep()

```
void Lora::sleep ( )
```

Set the LoRa module to sleep mode

8.19.3.12 tx_done_cb()

```
void Lora::tx_done_cb ( ) [private]
```

Callback function which is called as soon as a LoRa packet is fully transmitted

8.19.3.13 tx_timeout_cb()

```
void Lora::tx_timeout_cb ( ) [private]
```

Callback function which is called if a transmit timeout occurs

8.19.4 Member Data Documentation

8.19.4.1 _event_flags

```
rtos::EventFlags Lora::_event_flags [private]
```

Event flags used for event handling within this [Lora](#) class

8.19.4.2 _radio

```
SX126X_LoRaRadio Lora::_radio [private]
```

Radio object to control the LoRa module via the lower level driver from mbed OS.

8.19.4.3 _radio_callbacks

```
radio_events_t Lora::_radio_callbacks [private]
```

Struct holding the callbacks for the [_radio](#)

8.19.4.4 _rx_timeout

```
mbed::LowPowerTimeout Lora::_rx_timeout [private]
```

Receive timeout object used in the [receive\(\)](#) function

8.19.4.5 rx_payload

```
uint8_t Lora::rx_payload[MAX_DATA_BUFFER_SIZE_SX126X]
```

Data buffer for the last packet received

8.19.4.6 rx_payload_len

```
uint16_t Lora::rx_payload_len = 0
```

Size of the last packet received

8.19.4.7 rx_rssi

```
int8_t Lora::rx_rssi = 0
```

Received Signal Strength Indicator (RSSI) for the last packet received

8.19.4.8 rx_snr

```
int8_t Lora::rx_snr = 0
```

Signal to Noise Ratio (SNR) for the last packet received

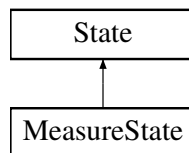
The documentation for this class was generated from the following files:

- src/driver/[Lora.h](#)
- src/driver/[Lora.cpp](#)

8.20 MeasureState Class Reference

```
#include <MeasureState.h>
```

Inheritance diagram for MeasureState:



Classes

- struct [avg_data_s](#)

Public Member Functions

- [MeasureState](#) ([LedIndicator](#) &led_indicator, [SDCard](#) &sd_card, [Communication](#) &communication, [SensorSPS30](#) &sps30, [SensorBME280](#) &bme280, rtos::EventFlags &event_flags)
- [~MeasureState](#) () override=default
- [State::Result](#) [handle](#) () override

Private Types

- enum class [SPS30State](#) { [OFF](#) , [STARTING](#) , [RUNNING](#) }
- typedef struct [MeasureState::avg_data_s](#) [avg_data_t](#)

Private Member Functions

- `mbed_error_status_t init_measurement_file ()`
- `void set_do_measurement_flag ()`
- `void set_create_new_file_flag ()`
- `void set_sps30_startup_done_flag ()`
- `State::Result run ()`
- `void send_measurement_data (const SensorSPS30::measurement_t &sps30_measurement, const SensorBME280::measurement_t &bme280_measurement)`
- `mbed_error_status_t store_measurement_data (const SensorSPS30::measurement_t &sps30_measurement, const SensorBME280::measurement_t &bme280_measurement)`
- `mbed_error_status_t get_sps30_info ()`

Private Attributes

- `SDCard & _sd_card`
- `Communication & _communication`
- `SensorSPS30 & _sps30`
- `SPS30State _sps30_state`
- `SensorBME280 & _bme280`
- `rtos::EventFlags & _event_flags`
- `std::string _data_file_name`
- `std::string _sps30_info`
- `mbed::LowPowerTicker _measurement_ticker`
- `mbed::LowPowerTicker _new_file_ticker`
- `mbed::LowPowerTimeout _sps30_startup_timeout`
- `std::vector< SensorSPS30::measurement_t > _sps30_data_buffer`
- `std::vector< avg_data_t > _avg_data`

Additional Inherited Members

8.20.1 Detailed Description

`State` handling the measurement process

8.20.2 Member Typedef Documentation

8.20.2.1 `avg_data_t`

```
typedef struct MeasureState::avg_data_s MeasureState::avg_data_t [private]
```

Struct used to store the measurement points for the moving average. Overloaded operators are used to calculate the average.

8.20.3 Member Enumeration Documentation

8.20.3.1 `SPS30State`

```
enum MeasureState::SPS30State [strong], [private]
```

Enum defining the SPS30 states

Enumerator

OFF	
STARTING	
RUNNING	

8.20.4 Constructor & Destructor Documentation

8.20.4.1 MeasureState()

```
MeasureState::MeasureState (
    LedIndicator & led_indicator,
    SDCard & sd_card,
    Communication & communication,
    SensorSPS30 & sps30,
    SensorBME280 & bme280,
    rtos::EventFlags & event_flags )
```

Create a [MeasureState](#)

Parameters

<i>led_indicator</i>	Reference to the LedIndicator to set the LED behavior
<i>sd_card</i>	Reference to the SDCard to write the measurement files to
<i>communication</i>	Reference to the Communication object to send measurement data via LoRa
<i>sps30</i>	Reference to the SensorSPS30
<i>bme280</i>	Reference to the SensorBME280
<i>event_flags</i>	Reference to the control event flags to react on occurring control events.

8.20.4.2 ~MeasureState()

```
MeasureState::~MeasureState ( ) [override], [default]
```

Default destructor

8.20.5 Member Function Documentation

8.20.5.1 `get_sps30_info()`

```
mbed_error_status_t MeasureState::get_sps30_info ( ) [private]
```

Retrieve the version and serial number from the SPS30 sensor and store it to `_sps30_info`.

Returns

MBED_SUCCESS on success, an `mbed_error_status_t` otherwise

8.20.5.2 `handle()`

```
State::Result MeasureState::handle ( ) [override], [virtual]
```

Start `_measurement_ticker` and `_new_file_ticker` to signalize when a measurement needs to be done or a new measurement file needs to be created. Init measurement file and the `_sps30_state` and clear the `_event_flags`. Call `run()` to execute the measurement process.

Returns

the returned value of `run()`

Implements `State`.

8.20.5.3 `init_measurement_file()`

```
mbed_error_status_t MeasureState::init_measurement_file ( ) [private]
```

Create a new measurement file and add the file header to the newly created file.

Returns

MBED_SUCCESS on success, otherwise an `mbed_error_status_t`

8.20.5.4 `run()`

```
State::Result MeasureState::run ( ) [private]
```

Handle the measurement process by reacting on set `_event_flags`.

Depending on the flag, which is set, execute one of the following paths:

- 1) `FLAG_BUTTON_MEASURE`: Call `set_do_measurement_flag()`
- 2) `FLAG_BUTTON_COMMAND`: Return with `State::Result::COMMAND_PRESSED`
- 3) `FLAG_BUTTON_STATUS`: Show the program status and battery level with `_led_indicator`
- 4) `FLAG_DO_MEASUREMENT`: Start SPS30 if it is not already starting or running and set `FLAG_SPS30_STARTUP_DONE` after a given start up timeout. If the SPS30 is already running, set `FLAG_SPS30_STARTUP_DONE` immediately.
- 5) `FLAG_SPS30_STARTUP_DONE`: Read the measurement data from the BME280 and the SPS30. Stop the SPS30 if the next measurement does not start within the next 5 seconds, otherwise keep the SPS30 running. Then write the measurement data to the SD card and send it via LoRa.
- 6) `FLAG_CREATE_NEW_FILE`: Create a new measurement file with `init_measurement_file()`

Returns

a `State::Result` depending on the return reason

8.20.5.5 send_measurement_data()

```
void MeasureState::send_measurement_data (
    const SensorSPS30::measurement\_t & sps30_measurement,
    const SensorBME280::measurement\_t & bme280_measurement ) [private]
```

Scale `sps30_measurement` values and `bme280_measurement` values to match the protocol. Send them to the LoRa communication thread, which will send the data to the gateway via LoRa.

Parameters

<code>sps30_measurement</code>	Measurement data of the SPS30 sensor
<code>bme280_measurement</code>	Measurement data of the BME280 sensor

8.20.5.6 set_create_new_file_flag()

```
void MeasureState::set_create_new_file_flag ( ) [private]
```

Set [FLAG_CREATE_NEW_FILE](#) in `_event_flags`

8.20.5.7 set_do_measurement_flag()

```
void MeasureState::set_do_measurement_flag ( ) [private]
```

Set [FLAG_DO_MEASUREMENT](#) in `_event_flags`

8.20.5.8 set_sps30_startup_done_flag()

```
void MeasureState::set_sps30_startup_done_flag ( ) [private]
```

Set [FLAG_SPS30_STARTUP_DONE](#) in `_event_flags`

8.20.5.9 store_measurement_data()

```
mbed_error_status_t MeasureState::store_measurement_data (
    const SensorSPS30::measurement\_t & sps30_measurement,
    const SensorBME280::measurement\_t & bme280_measurement ) [private]
```

Store the `sps30_measurement` data and `bme280_measurement` data on the SD card.

Parameters

<code>sps30_measurement</code>	Measurement data of the SPS30 sensor
<code>bme280_measurement</code>	Measurement data of the BME280 sensor

Returns

MBED_SUCCESS if write succeeded or SD card logging is turned off, an `mbed_error_status_t` otherwise.

8.20.6 Member Data Documentation

8.20.6.1 `_avg_data`

```
std::vector<avg_data_t> MeasureState::_avg_data [private]
```

Vector storing the latest measurement values for the moving average computation.

8.20.6.2 `_bme280`

```
SensorBME280& MeasureState::_bme280 [private]
```

Reference to the [SensorBME280](#) object

8.20.6.3 `_communication`

```
Communication& MeasureState::_communication [private]
```

Reference to the [Communication](#) handling the LoRa communication.

8.20.6.4 `_data_file_name`

```
std::string MeasureState::_data_file_name [private]
```

Timestamp when the currently used measurement file was created

8.20.6.5 `_event_flags`

```
rtos::EventFlags& MeasureState::_event_flags [private]
```

Reference to the control event flags used to control the measurement process

8.20.6.6 `_measurement_ticker`

```
mbed::LowPowerTicker MeasureState::_measurement_ticker [private]
```

Ticker signaling when it is time to do a measurement

8.20.6.7 `_new_file_ticker`

```
mbed::LowPowerTicker MeasureState::_new_file_ticker [private]
```

Ticker signaling when it is time to create a new measurement file

8.20.6.8 `_sd_card`

```
SDCard& MeasureState::_sd_card [private]
```

Reference to the [SDCard](#) to write the measurement files to

8.20.6.9 `_sps30`

```
SensorSPS30& MeasureState::_sps30 [private]
```

Reference to the [SensorSPS30](#) object

8.20.6.10 `_sps30_data_buffer`

```
std::vector<SensorSPS30::measurement_t> MeasureState::_sps30_data_buffer [private]
```

Vector storing the latest SPS30 measurement values for the moving average computation.

8.20.6.11 `_sps30_info`

```
std::string MeasureState::_sps30_info [private]
```

Version and serial numbers of the SPS30

8.20.6.12 `_sps30_startup_timeout`

```
mbed::LowPowerTimeout MeasureState::_sps30_startup_timeout [private]
```

Timeout used to wait for the SPS30 to start up

8.20.6.13 `_sps30_state`

```
SPS30State MeasureState::_sps30_state [private]
```

The current state of [_sps30](#)

The documentation for this class was generated from the following files:

- [src/state/MeasureState.h](#)
- [src/state/MeasureState.cpp](#)

8.21 Protocol Class Reference

```
#include <Protocol.h>
```

Public Member Functions

- struct [__attribute__](#) ((__packed__))
- struct [__attribute__](#) ((__packed__))
- [Protocol](#) (Lora &lora)
- void [send_auth](#) ()
- bool [is_auth_done](#) ()
- void [auth_reset](#) ()
- void [send_measured_data](#) (const [measured_data_t](#) &data)
- void [send_status_data](#) (const [status_data_t](#) &data)
- mbed_error_status_t [read_received](#) ()

Static Public Member Functions

- static uint32_t [get_sw_uuid](#) ()
- static uint32_t [get_hw_uuid](#) ()

Public Attributes

- [measured_data_t](#)
- [status_data_t](#)

Private Member Functions

- struct [__attribute__](#) ((__packed__))
- struct [__attribute__](#) ((__packed__))
- struct [__attribute__](#) ((__packed__))
- struct [__attribute__](#) ((__packed__))
- struct [__attribute__](#) ((__packed__)) base_ack_s
- mbed_error_status_t [crc_check](#) (const uint8_t *buf, uint16_t size)
- mbed_error_status_t [check_auth](#) (const uint8_t *buf, uint16_t size)
- mbed_error_status_t [check_ack](#) (const uint8_t *buf, uint16_t size)

Private Attributes

- [node_auth_t](#)
- [base_auth_t](#)
- [node_measured_data_t](#)
- [node_status_data_t](#)
- [base_ack_t](#)
- [Lora](#) & [_lora](#)
- uint16_t [_node_crc](#) = 0
- uint16_t [_base_crc](#) = 0
- uint8_t [_node_id](#) = 0
- bool [_client_auth_done](#) = false

8.21.1 Detailed Description

Implements the communication protocol between a sensor node and a smartmote gateway.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 Protocol()

```
Protocol::Protocol (
    Lora & lora ) [explicit]
```

Creates a [Protocol](#) object which uses `lora` to exchange packets with the gateway.

Parameters

<code>lora</code>	Reference to the Lora object acting as a driver for the LoRa module
-------------------	-------------------------------------------------------------------------------------

8.21.3 Member Function Documentation

8.21.3.1 __attribute__() [1/7]

```
struct Protocol::__attribute__ (
    (__packed__ ) [inline]
```

Struct defining the format for the measurement data within a [Protocol::node_measured_data_t](#) packet

8.21.3.2 __attribute__() [2/7]

```
struct Protocol::__attribute__ (
    (__packed__ ) [inline]
```

Struct defining the format for the status data within a [Protocol::node_status_data_t](#) packet

8.21.3.3 __attribute__() [3/7]

```
struct Protocol::__attribute__ (
    (__packed__ ) [inline], [private]
```

Struct defining the format for a node authentication request packet

8.21.3.4 __attribute__() [4/7]

```
struct Protocol::__attribute__ (
    (__packed__ ) [inline], [private]
```

Struct defining the format for the base station authentication response packet

8.21.3.5 __attribute__() [5/7]

```
struct Protocol::__attribute__ (
    (__packed__ ) [inline], [private]
```

Struct defining the format for the measurement data packet sent by the node

8.21.3.6 __attribute__() [6/7]

```
struct Protocol::__attribute__ (
    (__packed__ ) [inline], [private]
```

Struct defining the format for the status data packet sent by the node

8.21.3.7 __attribute__() [7/7]

```
struct Protocol::__attribute__ (
    (__packed__ ) [inline], [private]
```

Struct defining the format for the base station acknowledge response on a data packet

8.21.3.8 auth_reset()

```
void Protocol::auth_reset ( )
```

Resets the connection by setting [_client_auth_done](#) to false.

8.21.3.9 check_ack()

```
mbed_error_status_t Protocol::check_ack (
    const uint8_t * buf,
    uint16_t size ) [private]
```

Checks the response from the gateway on a sent data packet. After checking the length, the CRC checksum, the [_node_id](#) and the acknowledge byte are checked.

Parameters

<i>buf</i>	The buffer with the acknowledge packet
<i>size</i>	The size of the buffer

Returns

MBED_SUCCESS if the a valid acknowledge was received, an `mbed_error_status_t` otherwise

8.21.3.10 check_auth()

```
mbed_error_status_t Protocol::check_auth (
    const uint8_t * buf,
    uint16_t size ) [private]
```

Checks the authentication response from the gateway. First, the length is checked, then the `_base_crc` is extracted and if the `crc_check()` was successful, the `_node_id` is extracted and `_client_auth_done` is set to true.

Parameters

<i>buf</i>	The buffer with the received authentication response
<i>size</i>	The size of the buffer

Returns

MBED_SUCCESS if the authentication succeeded, MBED_ERROR_INVALID_SIZE or MBED_ERROR_CRC_ERROR otherwise

8.21.3.11 crc_check()

```
mbed_error_status_t Protocol::crc_check (
    const uint8_t * buf,
    uint16_t size ) [private]
```

Check if the CRC checksum at the end of `buf` is correct.

Parameters

<i>buf</i>	The buffer with a received packet
<i>size</i>	The size of <code>buf</code>

Returns

MBED_SUCCESS if the CRC checksum is correct, MBED_ERROR_CRC_ERROR otherwise

8.21.3.12 get_hw_uuid()

```
uint32_t Protocol::get_hw_uuid ( ) [static]
```

Calculate the hardware UUID, which is the 32-bit CRC checksum from the unique identifier of the microcontroller.

Returns

the 32-bit hardware UUID

8.21.3.13 get_sw_uuid()

```
uint32_t Protocol::get_sw_uuid ( ) [static]
```

Calculate the software UUID, which is the 32-bit CRC checksum of the app version string.

Returns

the 32-bit software UUID

8.21.3.14 is_auth_done()

```
bool Protocol::is_auth_done ( )
```

Returns

[_client_auth_done](#), which is true if the node is authenticated and false otherwise

8.21.3.15 read_received()

```
mbed_error_status_t Protocol::read_received ( )
```

Retrieve the received packets from [_lora](#) and check the validity of the packet with [check_auth\(\)](#) or [check_ack\(\)](#)

Returns

MBED_SUCESS if a valid authentication responds or a valid acknowledge was received and a [mbed_error↵_status_t](#) otherwise

8.21.3.16 send_auth()

```
void Protocol::send_auth ( )
```

Send an authentication message to the gateway.

8.21.3.17 send_measured_data()

```
void Protocol::send_measured_data (
    const measured\_data\_t & data )
```

Send the given measured `data` in format of [Protocol::node_measured_data_t](#) via LoRa to the gateway

Parameters

<i>data</i>	measured data to be sent
-------------	--------------------------

8.21.3.18 send_status_data()

```
void Protocol::send_status_data (
    const status\_data\_t & data )
```

Send the given status data in format of [Protocol::node_status_data_t](#) via LoRa to the gateway

Parameters

<i>data</i>	status data to be sent
-------------	------------------------

8.21.4 Member Data Documentation**8.21.4.1 _base_crc**

```
uint16_t Protocol::_base_crc = 0 [private]
```

The base CRC used to ensure message integrity

8.21.4.2 _client_auth_done

```
bool Protocol::_client_auth_done = false [private]
```

Authentication status

8.21.4.3 _lora

```
Lora& Protocol::_lora [private]
```

Reference to the [Lora](#) object used to control the LoRa module

8.21.4.4 _node_crc

```
uint16_t Protocol::_node_crc = 0 [private]
```

The node CRC used to ensure message integrity

8.21.4.5 `_node_id`

```
uint8_t Protocol::_node_id = 0 [private]
```

The node ID assigned by the gateway

8.21.4.6 `base_ack_t`

```
Protocol::base_ack_t [private]
```

8.21.4.7 `base_auth_t`

```
Protocol::base_auth_t [private]
```

8.21.4.8 `measured_data_t`

```
Protocol::measured_data_t
```

8.21.4.9 `node_auth_t`

```
Protocol::node_auth_t [private]
```

8.21.4.10 `node_measured_data_t`

```
Protocol::node_measured_data_t [private]
```

8.21.4.11 `node_status_data_t`

```
Protocol::node_status_data_t [private]
```

8.21.4.12 status_data_t

Protocol::status_data_t

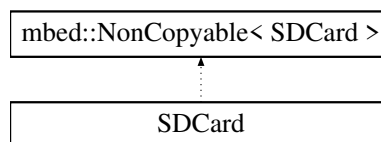
The documentation for this class was generated from the following files:

- src/communication/[Protocol.h](#)
- src/communication/[Protocol.cpp](#)

8.22 SDCard Class Reference

```
#include <SDCard.h>
```

Inheritance diagram for SDCard:



Public Member Functions

- [SDCard](#) (SDBlockDevice &sd_block_device, FATFileSystem &fat_file_system)
- mbed_error_status_t [init](#) ()
- mbed_error_status_t [deinit](#) ()
- mbed_error_status_t [format](#) ()
- mbed_error_status_t [write](#) (const std::string &filename, const std::string &data)
- mbed_error_status_t [check_available_memory](#) ()
- bool [is_full](#) ()

Private Member Functions

- bool [is_inserted](#) ()

Private Attributes

- bool [_is_full](#) = false
- mbed::DigitalIn [_n_detect](#)
- SDBlockDevice & [_sd_block_device](#)
- FATFileSystem & [_fat_file_system](#)
- bool [_initialized](#)
- PlatformMutex [_sd_card_mutex](#)

8.22.1 Detailed Description

Class handling the SD card operations

8.22.2 Constructor & Destructor Documentation

8.22.2.1 SDCard()

```
SDCard::SDCard (
    SDBlockDevice & sd_block_device,
    FATFileSystem & fat_file_system )
```

Create a [SDCard](#) object with the given `sd_block_device` and `fat_file_system`. Because the initialization of the [SDCard](#) can fail, it is done in the [init\(\)](#) function instead of the constructor.

Parameters

<code><i>sd_block_device</i></code>	reference to the SDBlockDevice object
<code><i>fat_file_system</i></code>	reference to the FATFileSystem object

8.22.3 Member Function Documentation

8.22.3.1 check_available_memory()

```
mbed_error_status_t SDCard::check_available_memory ( )
```

Check the available free memory on the SD card. If the SD card is full, [_is_full](#) is set to true.

Returns

MBED_SUCCESS if memory statistics could be read and the available memory is more than 1MB. Otherwise an `mbed_error_status_t` is returned.

8.22.3.2 deinit()

```
mbed_error_status_t SDCard::deinit ( )
```

Deinitializes the [SDCard](#) object by unmounting [_fat_file_system](#) and deinitializing [_sd_block_device](#).

Returns

MBED_SUCCESS on success, an `mbed_error_status_t` otherwise

8.22.3.3 format()

```
mbed_error_status_t SDCard::format ( )
```

Formats the SD card with a cluster size of 4kB

Returns

MBED_SUCCESS on success, an `mbed_error_status_t` otherwise

8.22.3.4 init()

```
mbed_error_status_t SDCard::init ( )
```

Checks if an SD card is detected, initializes the [_sd_block_device](#), sets its SPI frequency, mounts the [_fat_file_system](#) and checks if more than 1MB free memory is available on the SD card.

Returns

MBED_SUCCESS if initialization succeeded or [SDCard](#) is already initialized, an `mbed_error_status_t` otherwise

8.22.3.5 is_full()

```
bool SDCard::is_full ( )
```

Returns

[_is_full](#)

8.22.3.6 is_inserted()

```
bool SDCard::is_inserted ( ) [private]
```

Returns

true if an SD card is inserted/detected

8.22.3.7 write()

```
mbed_error_status_t SDCard::write (
    const std::string & filename,
    const std::string & data )
```

Writes the given `data` to the file with the given `filename` in append mode. If the file with the `filename` does not exist, it is created. The file is opened before the write operation and closed afterwards. After closing the file, the available free memory is checked.

Parameters

<i>filename</i>	the name of the file to which the <code>data</code> should be appended
<i>data</i>	data string that should be written to the specified file

Returns

MBED_SUCCESS on success, an `mbed_error_status_t` otherwise

8.22.4 Member Data Documentation

8.22.4.1 `_fat_file_system`

```
FATFileSystem& SDCard::_fat_file_system [private]
```

Reference to the `FATFileSystem` object

8.22.4.2 `_initialized`

```
bool SDCard::_initialized [private]
```

Flag which is true if the `SDCard` object is initialized

8.22.4.3 `_is_full`

```
bool SDCard::_is_full = false [private]
```

Flag which is set to true by `check_available_memory()` as soon as the SD card is full.

8.22.4.4 `_n_detect`

```
mbed::DigitalIn SDCard::_n_detect [private]
```

8.22.4.5 `_sd_block_device`

```
SDBlockDevice& SDCard::_sd_block_device [private]
```

Reference to the `SDBlockDevice` object

8.22.4.6 `_sd_card_mutex`

```
PlatformMutex SDCard::_sd_card_mutex [private]
```

Mutex to ensure sequential write access to the SD card, because write is called from different threads.

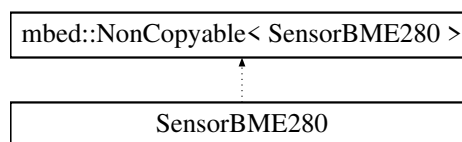
The documentation for this class was generated from the following files:

- [src/driver/SDCard.h](#)
- [src/driver/SDCard.cpp](#)

8.23 SensorBME280 Class Reference

```
#include <SensorBME280.h>
```

Inheritance diagram for SensorBME280:



Classes

- struct [bme280_handle_t](#)

Public Types

- typedef struct bme280_data [measurement_t](#)

Public Member Functions

- [SensorBME280](#) ()
- [~SensorBME280](#) ()=default
- mbed_error_status_t [init](#) ()
- mbed_error_status_t [read_measurement](#) ([measurement_t](#) *data)

Private Member Functions

- void [bme280_spi_init](#) ()
- int8_t [read_sensor_data](#) ([measurement_t](#) *data)
- void [correct_data](#) ([measurement_t](#) *data)

Static Private Member Functions

- static void [bme280_delay_us](#) (uint32_t period, void *intf_ptr)
- static int8_t [bme280_spi_read](#) (uint8_t reg_addr, uint8_t *reg_data, uint32_t len, void *intf_ptr)
- static int8_t [bme280_spi_write](#) (uint8_t reg_addr, const uint8_t *reg_data, uint32_t len, void *intf_ptr)

Private Attributes

- mbed::DigitalOut [_bme280_cs](#)
- mbed::SPI [_spi](#)
- struct bme280_dev [_dev](#)
- [bme280_handle_t](#) [_bme280_handle](#)

8.23.1 Detailed Description

Adapter class providing functions to control the Bosch BME280 environmental sensor

8.23.2 Member Typedef Documentation

8.23.2.1 measurement_t

```
typedef struct bme280_data SensorBME280::measurement\_t
```

Data structure for BME280 measurement data

8.23.3 Constructor & Destructor Documentation

8.23.3.1 SensorBME280()

```
SensorBME280::SensorBME280 ( )
```

Creates a [SensorBME280](#) object

8.23.3.2 ~SensorBME280()

```
SensorBME280::~~SensorBME280 ( ) [default]
```

Default destructor

8.23.4 Member Function Documentation

8.23.4.1 bme280_delay_us()

```
void SensorBME280::bme280\_delay\_us (
    uint32_t period,
    void * intf_ptr ) [static], [private]
```

Delay function for the BME280 manufacturer driver

Parameters

<i>period</i>	Time to wait in Microseconds
<i>intf_ptr</i>	Pointer to _bme280_handle (not used in this function)

8.23.4.2 bme280_spi_init()

```
void SensorBME280::bme280_spi_init ( ) [private]
```

Initialize the SPI for the BME280

8.23.4.3 bme280_spi_read()

```
int8_t SensorBME280::bme280_spi_read (
    uint8_t reg_addr,
    uint8_t * reg_data,
    uint32_t len,
    void * intf_ptr ) [static], [private]
```

SPI read function for the BME280 manufacturer driver

Parameters

<i>reg_addr</i>	BME280 register address to read from
<i>reg_data</i>	Buffer where the read data is written to
<i>len</i>	Number of bytes to read
<i>intf_ptr</i>	Pointer to _bme280_handle

Returns

always BME280_OK (defined in bme280_defs.h)

8.23.4.4 bme280_spi_write()

```
int8_t SensorBME280::bme280_spi_write (
    uint8_t reg_addr,
    const uint8_t * reg_data,
    uint32_t len,
    void * intf_ptr ) [static], [private]
```

SPI write function for the BME280 manufacturer driver

Parameters

<i>reg_addr</i>	BME280 register address to write to
<i>reg_data</i>	Data which should be written
<i>len</i>	Number of bytes to write
<i>intf_ptr</i>	Pointer to _bme280_handle

Returns

always BME280_OK (defined in bme280_defs.h)

8.23.4.5 correct_data()

```
void SensorBME280::correct_data (
    measurement_t * data ) [private]
```

Apply a correction offset and factor to the measurement *data*. The correction values are defined in [Config::global_config](#).

Parameters

<i>data</i>	the measurement data to be corrected
-------------	--------------------------------------

8.23.4.6 init()

```
mbed_error_status_t SensorBME280::init ( )
```

Initialization function for the BME280 and the SPI bus.

Returns

MBED_SUCCESS if initialization is successful, MBED_ERROR_INITIALIZATION_FAILED otherwise

8.23.4.7 read_measurement()

```
mbed_error_status_t SensorBME280::read_measurement (
    measurement_t * data )
```

Reads measurement data from BME280 and stores it into *data*

Parameters

<i>data</i>	where the measurement data is stored to
-------------	-----------------------------------------

Returns

MBED_SUCCESS if read operation succeeded, MBED_ERROR_READ_FAILED otherwise

8.23.4.8 read_sensor_data()

```
int8_t SensorBME280::read_sensor_data (
    measurement_t * data ) [private]
```

Read temperature, pressure and humidity from the sensor in forced mode.

Parameters

<i>data</i>	Buffer to write the measured data to
-------------	--------------------------------------

Returns

BME280_OK on success, a negative error code (see bme280_defs.h) otherwise

8.23.5 Member Data Documentation**8.23.5.1 _bme280_cs**

```
mbed::DigitalOut SensorBME280::_bme280_cs [private]
```

BME280 SPI chip select output

8.23.5.2 _bme280_handle

```
bme280_handle_t SensorBME280::_bme280_handle [private]
```

Handle for the bme280_spi_* functions

8.23.5.3 _dev

```
struct bme280_dev SensorBME280::_dev [private]
```

BME280 device structure for the manufacturer driver

8.23.5.4 `_spi`

```
mbd::SPI SensorBME280::_spi [private]
```

SPI object for the communication with the BME280 sensor

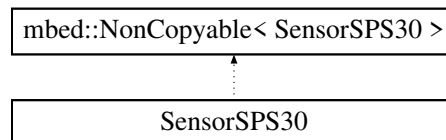
The documentation for this class was generated from the following files:

- `src/driver/SensorBME280.h`
- `src/driver/SensorBME280.cpp`

8.24 SensorSPS30 Class Reference

```
#include <SensorSPS30.h>
```

Inheritance diagram for SensorSPS30:



Public Types

- typedef struct sps30_measurement [measurement_t](#)

Public Member Functions

- [SensorSPS30](#) ()
- [~SensorSPS30](#) ()=default
- void [power_on](#) ()
- void [power_off](#) ()
- mbed_error_status_t [probe](#) ()
- mbed_error_status_t [get_serial](#) (std::string &serial)
- mbed_error_status_t [read_version](#) (sps30_version_information *version)
- mbed_error_status_t [start_measurement](#) ()
- void [stop_measurement](#) ()
- mbed_error_status_t [read_measurement](#) ([measurement_t](#) *measurement_data)
- mbed_error_status_t [start_manual_fan_cleaning](#) ()

Private Member Functions

- void [correct_data](#) ([measurement_t](#) *measurement_data)

Private Attributes

- mbed::DigitalOut [_supply_enable](#)
- mbed::BufferedSerial [_serial](#)

8.24.1 Detailed Description

Adapter class providing functions to control the Sensirion SPS30 particulate matter sensor.

8.24.2 Member Typedef Documentation

8.24.2.1 measurement_t

```
typedef struct sps30_measurement SensorSPS30::measurement\_t
```

Data structure for SPS30 measurement data

8.24.3 Constructor & Destructor Documentation

8.24.3.1 SensorSPS30()

```
SensorSPS30::SensorSPS30 ( )
```

Creates a [SensorSPS30](#) object

8.24.3.2 ~SensorSPS30()

```
SensorSPS30::~~SensorSPS30 ( ) [default]
```

Default destructor

8.24.4 Member Function Documentation

8.24.4.1 correct_data()

```
void SensorSPS30::correct\_data (  
    measurement\_t * measurement_data ) [private]
```

Correct the `measurement_data` with the correction values defined in [Config::global_config](#)

Parameters

<code>measurement_data</code>	the data to be corrected
-------------------------------	--------------------------

8.24.4.2 get_serial()

```
mbed_error_status_t SensorSPS30::get_serial (
    std::string & serial )
```

Read serial number string from SPS30

Parameters

<i>serial</i>	string to write the serial number to
---------------	--------------------------------------

Returns

MBED_SUCCESS on success, an mbed_error_status_t otherwise

8.24.4.3 power_off()

```
void SensorSPS30::power_off ( )
```

Turn power supply off for SPS30

8.24.4.4 power_on()

```
void SensorSPS30::power_on ( )
```

Turn power supply on for SPS30

8.24.4.5 probe()

```
mbed_error_status_t SensorSPS30::probe ( )
```

Probe SPS30

Returns

MBED_SUCCESS if probing was successful, MBED_ERROR_FAILED_OPERATION otherwise

8.24.4.6 read_measurement()

```
mbed_error_status_t SensorSPS30::read_measurement (
    measurement_t * measurement_data )
```

Read new measurement data from the SPS30 and correct the data with [correct_data\(\)](#).

Parameters

<i>measurement_data</i>	Struct to store the measured data to
-------------------------	--------------------------------------

Returns

MBED_SUCCESS on success, an `mbed_error_status_t` otherwise

8.24.4.7 read_version()

```
mbed_error_status_t SensorSPS30::read_version (
    sps30_version_information * version )
```

Read firmware, hardware and shdlc version number from SPS30

Parameters

<i>version</i>	struct to store the version numbers to
----------------	----------------------------------------

Returns

MBED_SUCCESS on success, an `mbed_error_status_t` otherwise

8.24.4.8 start_manual_fan_cleaning()

```
mbed_error_status_t SensorSPS30::start_manual_fan_cleaning ( )
```

Start a fan cleaning process which lasts about 10 seconds.

Returns

MBED_SUCCESS on success, MBED_ERROR_FAILED_OPERATION otherwise

8.24.4.9 start_measurement()

```
mbed_error_status_t SensorSPS30::start_measurement ( )
```

Set the SPS30 into measurement mode. The SPS30 is then continuously measuring.

Returns

MBED_SUCCESS on success, an `mbed_error_status_t` otherwise

8.24.4.10 stop_measurement()

```
void SensorSPS30::stop_measurement ( )
```

Let the SPS30 exit the measurement mode and turn off the power supply.

8.24.5 Member Data Documentation

8.24.5.1 _serial

```
mbd::BufferedSerial SensorSPS30::_serial [private]
```

Serial interface handler

8.24.5.2 _supply_enable

```
mbd::DigitalOut SensorSPS30::_supply_enable [private]
```

Digital output for the 5V supply enable

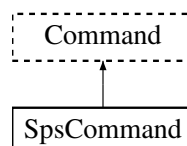
The documentation for this class was generated from the following files:

- [src/driver/SensorSPS30.h](#)
- [src/driver/SensorSPS30.cpp](#)

8.25 SpsCommand Class Reference

```
#include <SpsCommand.h>
```

Inheritance diagram for SpsCommand:



Public Member Functions

- [SpsCommand](#) (USBSerial &usb_serial, [SensorSPS30](#) &sps30)
- [~SpsCommand](#) () override=default

Private Member Functions

- void [execute_command](#) (std::map< std::string, std::string > ¶meter_map, std::vector< std::string > ¶meter_vec) override

Private Attributes

- [SensorSPS30](#) &_sps30

Additional Inherited Members

8.25.1 Detailed Description

Class handling commands to test and control the SPS30 sensor connected via UART serial interface

8.25.2 Constructor & Destructor Documentation

8.25.2.1 SpsCommand()

```
SpsCommand::SpsCommand (
    USBSerial & usb_serial,
    SensorSPS30 & sps30 )
```

Create a [SpsCommand](#) object

Parameters

<i>usb_serial</i>	reference to an USBSerial object
<i>sps30</i>	reference to the SensorSPS30 object which should be controlled

8.25.2.2 ~SpsCommand()

```
SpsCommand::~~SpsCommand ( ) [override], [default]
```

Default destructor

8.25.3 Member Function Documentation

8.25.3.1 execute_command()

```
void SpsCommand::execute_command (
    std::map< std::string, std::string > & parameter_map,
    std::vector< std::string > & parameter_vec ) [override], [private], [virtual]
```

Executes the sps command according to the given parameters. The usage of the command is described in the [sps_command_description](#).

Parameters

<i>parameter_map</i>	map holding the parameters that are given as key-value-pairs
<i>parameter_vec</i>	vector holding the option parameters

Implements [Command](#).

8.25.4 Member Data Documentation

8.25.4.1 _sps30

```
SensorSPS30& SpsCommand::_sps30 [private]
```

Reference to the [SensorSPS30](#) object used to control the sensor by calling its member functions.

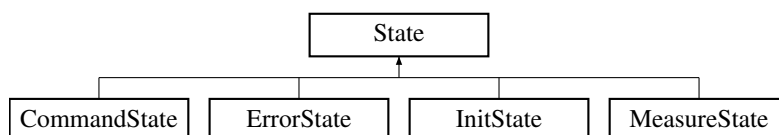
The documentation for this class was generated from the following files:

- [src/command/SpsCommand.h](#)
- [src/command/SpsCommand.cpp](#)

8.26 State Class Reference

```
#include <State.h>
```

Inheritance diagram for State:



Public Types

- enum class [Result](#) {
[SUCCESS](#) , [SENSOR_ERROR](#) , [SD_CARD_ERROR](#) , [FLASH_ERROR](#) ,
[LORA_ERROR](#) , [COMMAND_PRESSED](#) }

Public Member Functions

- [State](#) ([LedIndicator](#) & [_led_indicator](#))
- virtual [~State](#) ()=default
- virtual [State::Result](#) [handle](#) ()=0

Protected Attributes

- [LedIndicator](#) & [_led_indicator](#)

8.26.1 Detailed Description

Abstract base class for all states

8.26.2 Member Enumeration Documentation

8.26.2.1 Result

```
enum State::Result [strong]
```

Enum defining the possible return values for the states

Enumerator

SUCCESS	
SENSOR_ERROR	
SD_CARD_ERROR	
FLASH_ERROR	
LORA_ERROR	
COMMAND_PRESSED	

8.26.3 Constructor & Destructor Documentation

8.26.3.1 State()

```
State::State (  
    LedIndicator & \_led\_indicator ) [explicit]
```

Base constructor

Parameters

<code>_led_indicator</code>	Reference to the LedIndicator object
-----------------------------	------------------------------------------------------

8.26.3.2 ~State()

```
virtual State::~~State ( ) [virtual], [default]
```

Virtual default destructor

8.26.4 Member Function Documentation

8.26.4.1 handle()

```
virtual State::Result State::handle ( ) [pure virtual]
```

Pure virtual method handling the functions of the state. This method is called infinitely as long as the state is active.

Returns

result why the the handle function returned. This return value is used to decide which state is the next one

Implemented in [MeasureState](#), [InitState](#), [ErrorState](#), and [CommandState](#).

8.26.5 Member Data Documentation

8.26.5.1 _led_indicator

```
LedIndicator& State::_led_indicator [protected]
```

Reference to the [LedIndicator](#) object. Used to set the LEDs behavior according to the current state.

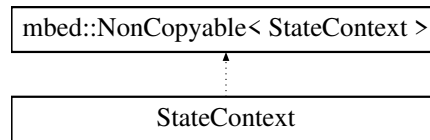
The documentation for this class was generated from the following files:

- [src/state/State.h](#)
- [src/state/State.cpp](#)

8.27 StateContext Class Reference

```
#include <StateContext.h>
```

Inheritance diagram for StateContext:



Public Member Functions

- [StateContext](#) ([SDCard](#) &sd_card, [Communication](#) &communication, [LedIndicator](#) &led_indicator, [CommandHandler](#) &command_handler, mbed::FlashIAP &flash, rtos::EventFlags &event_flags, [SensorSPS30](#) &sps30, [SensorBME280](#) &bme280, [Buttons](#) &buttons)
- [~StateContext](#) ()
- void [run](#) ()
- void [next_state](#) ([State::Result](#) result)

Private Attributes

- [State](#) * [_current_state](#)
- [InitState](#) * [_init_state](#)
- [MeasureState](#) * [_measure_state](#)
- [CommandState](#) * [_command_state](#)
- [ErrorState](#) * [_error_state](#)

8.27.1 Detailed Description

Class implementing a finite state machine, holding the [State](#) objects and handling the state transitions.

8.27.2 Constructor & Destructor Documentation

8.27.2.1 StateContext()

```

StateContext::StateContext (
    SDCard & sd_card,
    Communication & communication,
    LedIndicator & led_indicator,
    CommandHandler & command_handler,
    mbed::FlashIAP & flash,
    rtos::EventFlags & event_flags,
    SensorSPS30 & sps30,
    SensorBME280 & bme280,
    Buttons & buttons )
  
```

Create a [StateContext](#) Create an instance of every [State](#) subclass on the heap.

Parameters

<i>sd_card</i>	Reference to the SDCard to perform operations on the SD card
<i>communication</i>	Reference to the Communication object
<i>led_indicator</i>	Reference to the LedIndicator to set the LED behavior
<i>command_handler</i>	Reference to the CommandHandler handling commands over the USB serial COM port
<i>flash</i>	Reference to the FlashIAP to load and store configurations in the flash memory
<i>event_flags</i>	Reference to the event flags used to control the program flow
<i>sps30</i>	Reference to the SensorSPS30 object
<i>bme280</i>	Reference to the BME280 object
<i>buttons</i>	Reference to the Buttons object

8.27.2.2 ~StateContext()

```
StateContext::~StateContext ( )
```

Destructor, delete all [State](#) objects from the heap.

8.27.3 Member Function Documentation**8.27.3.1 next_state()**

```
void StateContext::next_state (
    State::Result result )
```

Depending *result* the next state is determined and the [_current_state](#) pointer is set to the next state.

Parameters

<i>result</i>	The State::Result of the _current_state , used to determine the next state
---------------	------------------------------------------------------------------------------------------------------------

8.27.3.2 run()

```
void StateContext::run ( )
```

Invoke [State::handle\(\)](#) of the [_current_state](#) and pass its return value to [next_state\(\)](#).

8.27.4 Member Data Documentation

8.27.4.1 `_command_state`

`CommandState*` `StateContext::_command_state` [private]

A pointer to the instance of this [State](#).

8.27.4.2 `_current_state`

`State*` `StateContext::_current_state` [private]

The current state

8.27.4.3 `_error_state`

`ErrorState*` `StateContext::_error_state` [private]

A pointer to the instance of this [State](#).

8.27.4.4 `_init_state`

`InitState*` `StateContext::_init_state` [private]

A pointer to the instance of this [State](#).

8.27.4.5 `_measure_state`

`MeasureState*` `StateContext::_measure_state` [private]

A pointer to the instance of this [State](#).

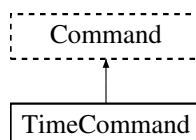
The documentation for this class was generated from the following files:

- `src/state/StateContext.h`
- `src/state/StateContext.cpp`

8.28 TimeCommand Class Reference

```
#include <TimeCommand.h>
```

Inheritance diagram for TimeCommand:



Public Member Functions

- [TimeCommand](#) (USBSerial &usb_serial)
- [~TimeCommand](#) () override=default

Private Member Functions

- void [execute_command](#) (std::map< std::string, std::string > ¶meter_map, std::vector< std::string > ¶meter_vec) override
- void [calibrate](#) ()
- void [wait_for_second_toggle](#) ()

Static Private Member Functions

- static time_t [parse_datetime](#) (const std::vector< std::string > ¶meters)

Private Attributes

- uint32_t [_cal_time_sec](#) = 30

Additional Inherited Members

8.28.1 Detailed Description

Class handling commands to set, print and calibrate the time and date

8.28.2 Constructor & Destructor Documentation

8.28.2.1 TimeCommand()

```
TimeCommand::TimeCommand (
    USBSerial & usb_serial ) [explicit]
```

Create a [TimeCommand](#) object

Parameters

<i>usb_serial</i>	reference to an USBSerial object
-------------------	----------------------------------

8.28.2.2 ~TimeCommand()

```
TimeCommand::~~TimeCommand ( ) [override], [default]
```

Default destructor

8.28.3 Member Function Documentation

8.28.3.1 calibrate()

```
void TimeCommand::calibrate ( ) [private]
```

Emits a 1Hz clock over [_usb_serial](#) by printing the number of passed seconds every second. It uses [wait_for_second_toggle\(\)](#) to wait until the RTC second counter changes. This function is executed in a separate highest priority thread to maximize the timing accuracy.

8.28.3.2 execute_command()

```
void TimeCommand::execute_command (
    std::map< std::string, std::string > & parameter_map,
    std::vector< std::string > & parameter_vec ) [override], [private], [virtual]
```

Executes the time command according to the given parameters. The usage of the command is described in the [time_command_description](#).

Parameters

<i>parameter_map</i>	map holding the parameters that are given as key-value-pairs
<i>parameter_vec</i>	vector holding the option parameters

Implements [Command](#).

8.28.3.3 parse_datetime()

```
time_t TimeCommand::parse_datetime (
    const std::vector< std::string > & parameters ) [static], [private]
```

Parses the date and time string into a struct tm and returns the corresponding time_t.

Parameters

<i>parameters</i>	vector with two strings, date and time
-------------------	----------------------------------------

Returns

timestamp of the parsed time as `time_t`

8.28.3.4 wait_for_second_toggle()

```
void TimeCommand::wait_for_second_toggle ( ) [private]
```

Repeatedly polls the current RTC time until the RTC time is incremented, which happens every second.

8.28.4 Member Data Documentation

8.28.4.1 _cal_time_sec

```
uint32_t TimeCommand::_cal_time_sec = 30 [private]
```

The calibration time in seconds that indicates how long `calibrate()` will be executed.

The documentation for this class was generated from the following files:

- [src/command/TimeCommand.h](#)
- [src/command/TimeCommand.cpp](#)

8.29 version_number_s Struct Reference

```
#include <defines.h>
```

Public Attributes

- `uint8_t` [major](#)
- `uint8_t` [minor](#)
- `uint16_t` [build](#)

8.29.1 Member Data Documentation

8.29.1.1 build

```
uint16_t version_number_s::build
```

8.29.1.2 major

```
uint8_t version_number_s::major
```

8.29.1.3 minor

```
uint8_t version_number_s::minor
```

The documentation for this struct was generated from the following file:

- src/defs/[defines.h](#)

Chapter 9

File Documentation

9.1 CHANGELOG.md File Reference

9.2 README.md File Reference

9.3 src/command/BmeCommand.cpp File Reference

```
#include "BmeCommand.h"  
#include "defines.h"
```

9.4 src/command/BmeCommand.h File Reference

```
#include "Command.h"  
#include "SensorBME280.h"
```

Classes

- class [BmeCommand](#)

9.5 src/command/Command.cpp File Reference

```
#include "Command.h"
```

9.6 src/command/Command.h File Reference

```
#include "USBSerial.h"  
#include <string>  
#include <vector>  
#include <map>
```

Classes

- class [Command](#)

9.7 src/command/CommandHandler.cpp File Reference

```
#include "CommandHandler.h"  
#include "TimeCommand.h"  
#include "ConfigCommand.h"  
#include "InfoCommand.h"  
#include "SpsCommand.h"  
#include "BmeCommand.h"  
#include "ThisThread.h"  
#include "Logger.h"
```

9.8 src/command/CommandHandler.h File Reference

```
#include "Command.h"  
#include "CircularBuffer.h"  
#include "EventFlags.h"  
#include "FlashIAP.h"  
#include "SensorSPS30.h"  
#include "SensorBME280.h"  
#include "defines.h"
```

Classes

- class [CommandHandler](#)

9.9 src/command/ConfigCommand.cpp File Reference

```
#include "ConfigCommand.h"  
#include "Config.h"  
#include "defines.h"
```

9.10 src/command/ConfigCommand.h File Reference

```
#include "Command.h"  
#include "FlashIAP.h"
```

Classes

- class [ConfigCommand](#)

9.11 src/command/InfoCommand.cpp File Reference

```
#include "InfoCommand.h"  
#include "Protocol.h"  
#include "defines.h"  
#include "utils.h"
```

9.12 src/command/InfoCommand.h File Reference

```
#include "Command.h"  
#include <string>
```

Classes

- class [InfoCommand](#)

9.13 src/command/SpsCommand.cpp File Reference

```
#include "SpsCommand.h"  
#include "defines.h"
```

9.14 src/command/SpsCommand.h File Reference

```
#include "Command.h"  
#include "SensorSPS30.h"
```

Classes

- class [SpsCommand](#)

9.15 src/command/TimeCommand.cpp File Reference

```
#include "TimeCommand.h"  
#include "defines.h"  
#include "Logger.h"  
#include "utils.h"  
#include "Thread.h"  
#include "mbed_rtc_time.h"
```

Macros

- #define [MS_PER_DAY](#) 86400000.0f

9.15.1 Macro Definition Documentation

9.15.1.1 MS_PER_DAY

```
#define MS_PER_DAY 86400000.0f
```

9.16 src/command/TimeCommand.h File Reference

```
#include "Command.h"
```

Classes

- class [TimeCommand](#)

9.17 src/communication/Communication.cpp File Reference

```
#include "Communication.h"  
#include "Battery.h"  
#include "defines.h"  
#include "Logger.h"
```

Macros

- #define [FLAG_SEND_MEASURED_DATA](#) (1U << 0U)
- #define [FLAG_SEND_STATUS_DATA](#) (1U << 1U)

9.17.1 Macro Definition Documentation

9.17.1.1 FLAG_SEND_MEASURED_DATA

```
#define FLAG_SEND_MEASURED_DATA (1U << 0U)
```

9.17.1.2 FLAG_SEND_STATUS_DATA

```
#define FLAG_SEND_STATUS_DATA (1U << 1U)
```

9.18 src/communication/Communication.h File Reference

```
#include "Lora.h"  
#include "Protocol.h"  
#include "LowPowerTicker.h"
```

Classes

- class [Communication](#)

9.19 src/communication/Protocol.cpp File Reference

```
#include "Protocol.h"  
#include "MbedCRC.h"  
#include "mbed_error.h"  
#include "mbed_debug.h"  
#include "utils.h"  
#include "Logger.h"
```

Macros

- #define [SMARTMOTE_AUTH_ID](#) 0xff
- #define [SMARTMOTE_HW_ID](#) 0x4220
- #define [SMARTMOTE_ACK](#) 0x5a
- #define [MEASURED_DATA_TYPE_ID](#) 32
- #define [STATUS_DATA_TYPE_ID](#) 39

9.19.1 Macro Definition Documentation

9.19.1.1 MEASURED_DATA_TYPE_ID

```
#define MEASURED_DATA_TYPE_ID 32
```

9.19.1.2 SMARTMOTE_ACK

```
#define SMARTMOTE_ACK 0x5a
```

9.19.1.3 SMARTMOTE_AUTH_ID

```
#define SMARTMOTE_AUTH_ID 0xff
```

9.19.1.4 SMARTMOTE_HW_ID

```
#define SMARTMOTE_HW_ID 0x4220
```

9.19.1.5 STATUS_DATA_TYPE_ID

```
#define STATUS_DATA_TYPE_ID 39
```

9.20 src/communication/Protocol.h File Reference

```
#include "Lora.h"
```

Classes

- class [Protocol](#)

9.21 src/config/Config.cpp File Reference

```
#include "Config.h"
#include <climits>
#include <cstring>
#include "Logger.h"
```

9.22 src/config/Config.h File Reference

```
#include "defines.h"
#include <vector>
#include <map>
#include "utils.h"
#include "mbed_error.h"
#include "FlashIAP.h"
```

Classes

- class [Config](#)
- union [Config::config_value_t](#)
- struct [Config::entry_t](#)

Macros

- `#define` [NUM_CONFIG_ENTRIES](#) 49

9.22.1 Macro Definition Documentation

9.22.1.1 NUM_CONFIG_ENTRIES

```
#define NUM_CONFIG_ENTRIES 49
```

Number of config entries in [Config::global_config](#)

9.23 src/defs/defines.h File Reference

```
#include <cstdint>
```

Classes

- struct [version_number_s](#)

Macros

- #define [VERSION_MAJOR](#) 0U
Major version.
- #define [VERSION_MINOR](#) 4U
Minor version.
- #define [VERSION_BUILD](#) 3U
Build version.
- #define [FLAG_BUTTON_MEASURE](#) (1U << 1U)
- #define [FLAG_BUTTON_STATUS](#) (1U << 2U)
- #define [FLAG_BUTTON_COMMAND](#) (1U << 3U)
- #define [FLAG_RECEIVED_COMMAND](#) (1U << 4U)
- #define [FLAG_DO_MEASUREMENT](#) (1U << 5U)
- #define [FLAG_SPS30_STARTUP_DONE](#) (1U << 6U)
- #define [FLAG_CREATE_NEW_FILE](#) (1U << 7U)
- #define [STORAGE_ADDR_CONFIG](#) POST_APPLICATION_ADDR
Flash address where the values of the [Config::global_config](#) are stored. Maximum size is FLASH_PAGE_SIZE (2kB).
- #define [BUTTON_DEBOUNCE_TIME](#) 50
Minimum time in milliseconds a button must be pressed to signal a short button press.
- #define [BUTTON_COMMAND_MIN_PRESS_TIME](#) 3000
Minimum time in milliseconds the command-button must be pressed to signal a long button press.
- #define [COMMAND_HANDLER_MAX_BUF_SIZE](#) 1024
Maximum buffer size for the USB serial interface.
- #define [LORA_RESEND_WAIT_MS](#) 10000
Time to wait before authentication request is sent again after a wrong response was received.
- #define [LORA_RX_TIMEOUT_MS](#) 5000
Maximum time to wait for a response after sending a request or data to the gateway.
- #define [MAX_SEND_DATA_ATTEMPTS](#) 3
Maximum number of attempts to send a data packet until a new authentication message is sent.
- #define [FLAG_LORA_TX_DONE](#) (1U << 0U)
Event flag which is set as soon as transmission is done.
- #define [FLAG_LORA_RX_DONE](#) (1U << 1U)
Event flag which is set as soon as a message is received.
- #define [FLAG_LORA_RX_TIMEOUT](#) (1U << 2U)
Event flag which is set if a receive timeout occurred.
- #define [BME280_TEST_STARTUP_TIME_MS](#) 1000
- #define [DATA_FILE_CREATION_PERIOD_SEC](#) 86400
Interval at which a new data file is created (24 hours)

Enumerations

- enum [LogLevel](#) {
 [LVL_SUPPRESS](#) = 0 , [LVL_ERROR](#) , [LVL_INFO](#) , [LVL_DEBUG](#) ,
 [NUM_LOG_LEVELS](#) }

Variables

- const struct [version_number_s](#) version = {[VERSION_MAJOR](#), [VERSION_MINOR](#), [VERSION_BUILD](#)}
- const char *const [time_command_name](#) = "time"
- const char *const [time_command_description](#)
- const char *const [config_command_name](#) = "config"
- const char *const [config_command_description](#)
- const char *const [sps_command_name](#) = "sps"
- const char *const [sps_command_description](#)
- const char *const [bme_command_name](#) = "bme"
- const char *const [bme_command_description](#)
- const char *const [info_command_name](#) = "info"
- const char *const [info_command_description](#)
- const char *const [log_level_names](#) []

9.23.1 Macro Definition Documentation

9.23.1.1 BME280_TEST_STARTUP_TIME_MS

```
#define BME280_TEST_STARTUP_TIME_MS 1000
```

9.23.1.2 BUTTON_COMMAND_MIN_PRESS_TIME

```
#define BUTTON_COMMAND_MIN_PRESS_TIME 3000
```

Minimum time in milliseconds the command-button must be pressed to signal a long button press.

9.23.1.3 BUTTON_DEBOUNCE_TIME

```
#define BUTTON_DEBOUNCE_TIME 50
```

Minimum time in milliseconds a button must be pressed to signal a short button press.

9.23.1.4 COMMAND_HANDLER_MAX_BUF_SIZE

```
#define COMMAND_HANDLER_MAX_BUF_SIZE 1024
```

Maximum buffer size for the USB serial interface.

9.23.1.5 DATA_FILE_CREATION_PERIOD_SEC

```
#define DATA_FILE_CREATION_PERIOD_SEC 86400
```

Interval at which a new data file is created (24 hours)

9.23.1.6 FLAG_BUTTON_COMMAND

```
#define FLAG_BUTTON_COMMAND (1U << 3U)
```

9.23.1.7 FLAG_BUTTON_MEASURE

```
#define FLAG_BUTTON_MEASURE (1U << 1U)
```

9.23.1.8 FLAG_BUTTON_STATUS

```
#define FLAG_BUTTON_STATUS (1U << 2U)
```

9.23.1.9 FLAG_CREATE_NEW_FILE

```
#define FLAG_CREATE_NEW_FILE (1U << 7U)
```

9.23.1.10 FLAG_DO_MEASUREMENT

```
#define FLAG_DO_MEASUREMENT (1U << 5U)
```

9.23.1.11 FLAG_LORA_RX_DONE

```
#define FLAG_LORA_RX_DONE (1U << 1U)
```

Event flag which is set as soon as a message is received.

9.23.1.12 FLAG_LORA_RX_TIMEOUT

```
#define FLAG_LORA_RX_TIMEOUT (1U << 2U)
```

Event flag which is set if a receive timeout occurred.

9.23.1.13 FLAG_LORA_TX_DONE

```
#define FLAG_LORA_TX_DONE (1U << 0U)
```

Event flag which is set as soon as transmission is done.

9.23.1.14 FLAG_RECEIVED_COMMAND

```
#define FLAG_RECEIVED_COMMAND (1U << 4U)
```

9.23.1.15 FLAG_SPS30_STARTUP_DONE

```
#define FLAG_SPS30_STARTUP_DONE (1U << 6U)
```

9.23.1.16 LORA_RESEND_WAIT_MS

```
#define LORA_RESEND_WAIT_MS 10000
```

Time to wait before authentication request is sent again after a wrong response was received.

9.23.1.17 LORA_RX_TIMEOUT_MS

```
#define LORA_RX_TIMEOUT_MS 5000
```

Maximum time to wait for a response after sending a request or data to the gateway.

9.23.1.18 MAX_SEND_DATA_ATTEMPTS

```
#define MAX_SEND_DATA_ATTEMPTS 3
```

Maximum number of attempts to send a data packet until a new authentication message is sent.

9.23.1.19 STORAGE_ADDR_CONFIG

```
#define STORAGE_ADDR_CONFIG POST_APPLICATION_ADDR
```

Flash address where the values of the [Config::global_config](#) are stored. Maximum size is FLASH_PAGE_SIZE (2kB).

9.23.1.20 VERSION_BUILD

```
#define VERSION_BUILD 3U
```

Build version.

9.23.1.21 VERSION_MAJOR

```
#define VERSION_MAJOR 0U
```

Major version.

9.23.1.22 VERSION_MINOR

```
#define VERSION_MINOR 4U
```

Minor version.

9.23.2 Enumeration Type Documentation

9.23.2.1 LogLevel

```
enum LogLevel
```

Enum defining the supported log levels.

Enumerator

LVL_SUPPRESS	
LVL_ERROR	
LVL_INFO	
LVL_DEBUG	
NUM_LOG_LEVELS	

9.23.3 Variable Documentation

9.23.3.1 bme_command_description

```
const char* const bme_command_description
```

Initial value:

```
=
    "Usage: bme COMMAND\n"
    "Commands:\n"
    "\tinit\tInitialize BME280\n"
    "\tread\tRead measurement values from sensor (Make sure init is called in advance)\n"
```

9.23.3.2 bme_command_name

```
const char* const bme_command_name = "bme"
```

9.23.3.3 config_command_description

```
const char* const config_command_description
```

Initial value:

```
=
    "Usage: config [OPTIONS]\n"
    "Options:\n"
    "\t<none>\t\tPrint current configuration\n"
    "\t<name>=<value>\tSet the configuration parameter <name> to the given <value>.\n"
    "\t\t\t\tMultiple parameters can be set in one command by seperating key-value-pairs with spaces.\n"
```

9.23.3.4 config_command_name

```
const char* const config_command_name = "config"
```

9.23.3.5 info_command_description

```
const char* const info_command_description
```

Initial value:

```
=  
    "Usage: info\n"  
    "Print software version and UUIDs\n"
```

9.23.3.6 info_command_name

```
const char* const info_command_name = "info"
```

9.23.3.7 log_level_names

```
const char* const log_level_names[]
```

Initial value:

```
= {  
    "SUPPR",  
    "ERROR",  
    "INFO ",  
    "DEBUG"  
}
```

Names of the supported log levels. The order of the names must be the same as in the [LogLevel](#) enum from above.

9.23.3.8 sps_command_description

```
const char* const sps_command_description
```

Initial value:

```
=  
    "Usage: sps COMMAND\n"  
    "Commands:\n"  
    "\ton\t\tEnable SPS30 power supply\n"  
    "\toff\t\tDisable SPS30 power supply\n"  
    "\tprobe\t\tPerform probe, to check if SPS30 is responding\n"  
    "\tserial\t\tPrint serial number\n"  
    "\tversion\t\tPrint version numbers for firmware, SHDL and hardware\n"  
    "\tstart\t\tStart measurement procedure\n"  
    "\tstop\t\tStop measurement procedure\n"  
    "\tread\t\tRead measurement values from sensor\n"  
    "\tclean\t\tStart fan cleaning (only works after \"sps start\")\n"
```

9.23.3.9 sps_command_name

```
const char* const sps_command_name = "sps"
```

9.23.3.10 time_command_description

```
const char* const time_command_description
```

Initial value:

```
=
"Usage: time [OPTIONS]\n"
"Options:\n"
"\t<none>\t\t\tPrint current time\n"
"\tset YYYY-MM-DD HH:MM:SS\t\tSet time in specified format\n"
"\ttick <num_ticks>\t\tPrint for <num_ticks> seconds every time, the RTC second counter toggles,\n"
"\t\t\t\tthe amount of passed seconds. Used to measure RTC time drift.\n"
"\tcalibrate <milliseconds>\tCalibrate the RTC by sending the time drift per day in milliseconds\n"
```

9.23.3.11 time_command_name

```
const char* const time_command_name = "time"
```

9.23.3.12 version

```
const struct version_number_s version = {VERSION_MAJOR, VERSION_MINOR, VERSION_BUILD}
```

9.24 src/driver/Battery.cpp File Reference

```
#include "Battery.h"
#include "Logger.h"
#include "AnalogIn.h"
#include "DigitalOut.h"
#include "analogin_api.h"
#include "mbed_error.h"
#include "ThisThread.h"
```

Functions

- mbed::DigitalOut [bat_level_en](#) (MBED_CONF_APP_BATTERY_LEVEL_EN, false)
- mbed::AnalogIn [bat_level_input](#) (MBED_CONF_APP_BATTERY_LEVEL)
- void [analogin_init_direct](#) (analogin_t *obj, const PinMap *pinmap)

9.24.1 Function Documentation

9.24.1.1 `analogin_init_direct()`

```
void analogin_init_direct (
    analogin_t * obj,
    const PinMap * pinmap )
```

Overwrite the default ADC configuration to increase oversampling ratio and thus increase the accuracy of the results.

9.24.1.2 `bat_level_en()`

```
mbed::DigitalOut bat_level_en (
    MBED_CONF_APP_BATTERY_LEVEL_EN ,
    false )
```

9.24.1.3 `bat_level_input()`

```
mbed::AnalogIn bat_level_input (
    MBED_CONF_APP_BATTERY_LEVEL )
```

9.25 `src/driver/Battery.h` File Reference

Classes

- class [Battery](#)

9.26 `src/driver/Buttons.cpp` File Reference

```
#include "Buttons.h"
#include "defines.h"
```

9.27 `src/driver/Buttons.h` File Reference

```
#include "EventFlags.h"
#include "InterruptIn.h"
#include "LowPowerTimer.h"
#include "NonCopyable.h"
```

Classes

- class [Buttons](#)

9.28 src/driver/LedIndicator.cpp File Reference

```
#include "LedIndicator.h"  
#include "Battery.h"  
#include "Logger.h"  
#include <chrono>
```

Macros

- `#define BATTERY_THRESHOLD_LED_1` 7.07
- `#define BATTERY_THRESHOLD_LED_2` 7.24
- `#define BATTERY_THRESHOLD_LED_3` 7.64

9.28.1 Macro Definition Documentation

9.28.1.1 BATTERY_THRESHOLD_LED_1

```
#define BATTERY_THRESHOLD_LED_1 7.07
```

9.28.1.2 BATTERY_THRESHOLD_LED_2

```
#define BATTERY_THRESHOLD_LED_2 7.24
```

9.28.1.3 BATTERY_THRESHOLD_LED_3

```
#define BATTERY_THRESHOLD_LED_3 7.64
```

9.29 src/driver/LedIndicator.h File Reference

```
#include "DigitalOut.h"  
#include "LowPowerTimeout.h"  
#include "NonCopyable.h"
```

Classes

- class [LedIndicator](#)

9.30 src/driver/Lora.cpp File Reference

```
#include "Lora.h"  
#include "defines.h"  
#include "Logger.h"  
#include "Config.h"
```

9.31 src/driver/Lora.h File Reference

```
#include "SX126X_LoRaRadio.h"  
#include "EventFlags.h"  
#include "LowPowerTimeout.h"  
#include "mbed_error.h"  
#include "NonCopyable.h"
```

Classes

- class [Lora](#)

Macros

- #define [LORA_CFG_MODEM](#) MODEM_LORA
- #define [LORA_CFG_TX_FDEV](#) 25000
- #define [LORA_CFG_RX_BANDWIDTH_AFC](#) 0
- #define [LORA_CFG_RX_PAYLOAD_LEN](#) 255

9.31.1 Macro Definition Documentation

9.31.1.1 LORA_CFG_MODEM

```
#define LORA_CFG_MODEM MODEM_LORA
```

9.31.1.2 LORA_CFG_RX_BANDWIDTH_AFC

```
#define LORA_CFG_RX_BANDWIDTH_AFC 0
```

9.31.1.3 LORA_CFG_RX_PAYLOAD_LEN

```
#define LORA_CFG_RX_PAYLOAD_LEN 255
```

9.31.1.4 LORA_CFG_TX_FDEV

```
#define LORA_CFG_TX_FDEV 25000
```

9.32 src/driver/SDCard.cpp File Reference

```
#include "SDCard.h"  
#include "mbed_debug.h"  
#include "utils.h"
```

9.33 src/driver/SDCard.h File Reference

```
#include "SDBlockDevice.h"  
#include "FATFileSystem.h"  
#include "PlatformMutex.h"  
#include "mbed_error.h"  
#include "DigitalIn.h"  
#include <string>
```

Classes

- class [SDCard](#)

9.34 src/driver/SensorBME280.cpp File Reference

```
#include "SensorBME280.h"  
#include "ThisThread.h"  
#include "Logger.h"  
#include "bme280.h"  
#include "bme280_defs.h"
```

9.35 src/driver/SensorBME280.h File Reference

```
#include "bme280.h"  
#include "mbed_error.h"  
#include "NonCopyable.h"  
#include "DigitalOut.h"  
#include "SPI.h"
```

Classes

- class [SensorBME280](#)
- struct [SensorBME280::bme280_handle_t](#)

9.36 src/driver/SensorSPS30.cpp File Reference

```
#include "SensorSPS30.h"  
#include "sensirion_uart.h"  
#include "ThisThread.h"  
#include "Logger.h"
```

9.37 src/driver/SensorSPS30.h File Reference

```
#include "sps30.h"  
#include "NonCopyable.h"  
#include "DigitalOut.h"  
#include "BufferedSerial.h"  
#include "mbed_error.h"
```

Classes

- class [SensorSPS30](#)

9.38 src/logging/Logger.cpp File Reference

```
#include "Logger.h"  
#include "utils.h"  
#include "mbed_debug.h"
```

9.39 src/logging/Logger.h File Reference

```
#include "SDCard.h"  
#include "defines.h"  
#include "Config.h"  
#include <string>  
#include <array>  
#include <map>  
#include <cstring>
```

Classes

- class [Logger](#)

Macros

- #define [__FILENAME__](#) (strchr(__FILE__, '/') ? strchr(__FILE__, '/') + 1 : __FILE__)
- #define [LOG](#)(lvl, fmt, args...) [Logger::log_internal](#)([__FILENAME__](#), lvl, fmt, ## args)

9.39.1 Macro Definition Documentation

9.39.1.1 [__FILENAME__](#)

```
#define __FILENAME__ (strchr(__FILE__, '/') ? strchr(__FILE__, '/') + 1 : __FILE__)
```

Macro to extract filename without file extension and path from **FILE**

9.39.1.2 [LOG](#)

```
#define LOG(  
    lvl,  
    fmt,  
    args... ) Logger::log\_internal(\_\_FILENAME\_\_, lvl, fmt, ## args)
```

Macro to create a log entry. The filename, where the log entry is created is added automatically.

9.40 src/main.cpp File Reference

```
#include "mbed.h"  
#include "LedIndicator.h"  
#include "SDCard.h"  
#include "Logger.h"  
#include "SensorSPS30.h"  
#include "SensorBME280.h"  
#include "Buttons.h"  
#include "StateContext.h"  
#include "Communication.h"
```

Functions

- int [main](#) ()

9.40.1 Function Documentation

9.40.1.1 main()

```
int main ( )
```

9.41 src/state/CommandState.cpp File Reference

```
#include "CommandState.h"  
#include "ThisThread.h"  
#include "Thread.h"  
#include "Logger.h"
```

9.42 src/state/CommandState.h File Reference

```
#include "State.h"  
#include "CommandHandler.h"  
#include "EventFlags.h"
```

Classes

- class [CommandState](#)

9.43 src/state/ErrorMessage.cpp File Reference

```
#include "ErrorMessage.h"  
#include "Logger.h"  
#include "ThisThread.h"
```

Functions

- void [mbed_error_reboot_callback](#) (mbed_error_ctx *error_context)

9.43.1 Function Documentation

9.43.1.1 mbed_error_reboot_callback()

```
void mbed_error_reboot_callback (
    mbed_error_ctx * error_context )
```

9.44 src/state/ErrorState.h File Reference

```
#include "State.h"
#include "EventFlags.h"
```

Classes

- class [ErrorState](#)

9.45 src/state/InitState.cpp File Reference

```
#include "InitState.h"
#include "Logger.h"
#include "utils.h"
#include "mbed_error.h"
#include "mbed_rtc_time.h"
```

Macros

- #define [TIMESTAMP_20200101](#) 1577836800

9.45.1 Macro Definition Documentation

9.45.1.1 TIMESTAMP_20200101

```
#define TIMESTAMP_20200101 1577836800
```

9.46 src/state/InitState.h File Reference

```
#include "State.h"
#include "SDCard.h"
#include "Communication.h"
#include "SensorSPS30.h"
#include "SensorBME280.h"
#include "Buttons.h"
#include "FlashIAP.h"
```

Classes

- class [InitState](#)

9.47 src/state/MeasureState.cpp File Reference

```
#include "MeasureState.h"
#include "Communication.h"
#include "Protocol.h"
#include "Logger.h"
#include "utils.h"
#include "Config.h"
```

9.48 src/state/MeasureState.h File Reference

```
#include "State.h"
#include "SDCard.h"
#include "Communication.h"
#include "SensorSPS30.h"
#include "SensorBME280.h"
#include "LowPowerTicker.h"
#include "EventFlags.h"
#include <vector>
```

Classes

- class [MeasureState](#)
- struct [MeasureState::avg_data_s](#)

9.49 src/state/State.cpp File Reference

```
#include "State.h"
```

9.50 src/state/State.h File Reference

```
#include "LedIndicator.h"
```

Classes

- class [State](#)

9.51 src/state/StateContext.cpp File Reference

```
#include "StateContext.h"
#include "CommandHandler.h"
#include "InitState.h"
#include "MeasureState.h"
#include "CommandState.h"
#include "ErrorState.h"
#include "mbed.h"
```

9.52 src/state/StateContext.h File Reference

```
#include "SensorSPS30.h"
#include "SensorBME280.h"
#include "Communication.h"
#include "SDCard.h"
#include "LedIndicator.h"
#include "CommandHandler.h"
#include "Config.h"
#include "USBSerial.h"
#include "EventFlags.h"
#include "NonCopyable.h"
#include "FlashIAP.h"
#include "Buttons.h"
#include "State.h"
#include "InitState.h"
#include "MeasureState.h"
#include "CommandState.h"
#include "ErrorState.h"
```

Classes

- class [StateContext](#)

9.53 src/utils/utils.cpp File Reference

```
#include "utils.h"
#include "defines.h"
#include "MbedCRC.h"
#include "trng_api.h"
```

Namespaces

- [utils](#)

Functions

- void [utils::get_formatted_time_string](#) (const char *format_string, std::string &dest, time_t timestamp)
- bool [utils::is_number](#) (const std::string &str)
- bool [utils::string_to_bool](#) (const std::string &str, bool &dest)
- std::string [utils::to_hex_string](#) (uint32_t number)
- std::string [utils::app_version](#) ()
- uint16_t [utils::calc_crc16](#) (const void *buf, int size)
- uint32_t [utils::calc_crc32](#) (const void *buf, int size)
- uint32_t [utils::get_rand](#) ()

9.54 src/utils/utils.h File Reference

```
#include <ctime>
#include <string>
```

Namespaces

- [utils](#)

Macros

- #define [REVERSE_2_BYTE](#)(n) (((n) & 0x00FFU) << 8U) | (((n) & 0xFF00U) >> 8U))
Swap least and most significant byte of a uint16_t.
- #define [TRY](#)(func)
Execute func and if it fails return with the return value of func.

Functions

- void [utils::get_formatted_time_string](#) (const char *format_string, std::string &dest, time_t timestamp)
- bool [utils::is_number](#) (const std::string &str)
- bool [utils::string_to_bool](#) (const std::string &str, bool &dest)
- std::string [utils::to_hex_string](#) (uint32_t number)
- std::string [utils::app_version](#) ()
- uint16_t [utils::calc_crc16](#) (const void *buf, int size)
- uint32_t [utils::calc_crc32](#) (const void *buf, int size)
- uint32_t [utils::get_rand](#) ()

9.54.1 Macro Definition Documentation

9.54.1.1 REVERSE_2_BYTE

```
#define REVERSE_2_BYTE(  
    n ) (((n) & 0x00FFU) << 8U) | (((n) & 0xFF00U) >> 8U)
```

Swap least and most significant byte of a uint16_t.

9.54.1.2 TRY

```
#define TRY(  
    func )
```

Value:

```
{  
    auto err = (func);  
    if (0 != err) {  
        return err;  
    }  
}
```

Execute func and if it fails return with the return value of func.

Index

- __FILENAME__
 - Logger.h, [139](#)
 - __attribute__
 - Protocol, [87](#), [88](#)
 - _avg_data
 - MeasureState, [84](#)
 - _base_crc
 - Protocol, [91](#)
 - _bat_led_1
 - LedIndicator, [69](#)
 - _bat_led_2
 - LedIndicator, [70](#)
 - _bat_led_3
 - LedIndicator, [70](#)
 - _bat_led_timeout
 - LedIndicator, [70](#)
 - _bme280
 - BmeCommand, [32](#)
 - InitState, [65](#)
 - MeasureState, [84](#)
 - _bme280_cs
 - SensorBME280, [101](#)
 - _bme280_handle
 - SensorBME280, [101](#)
 - _buffer
 - CommandHandler, [42](#)
 - _button_left
 - Buttons, [34](#)
 - _button_left_timer
 - Buttons, [34](#)
 - _button_right
 - Buttons, [34](#)
 - _button_right_timer
 - Buttons, [35](#)
 - _buttons
 - InitState, [65](#)
 - _cal_time_sec
 - TimeCommand, [116](#)
 - _client_auth_done
 - Protocol, [91](#)
 - _command_handler
 - CommandState, [45](#)
 - _command_state
 - StateContext, [112](#)
 - _commands
 - CommandHandler, [43](#)
 - _communication
 - InitState, [65](#)
 - MeasureState, [84](#)
 - _control_event_flags
 - ErrorState, [61](#)
 - _current_state
 - StateContext, [113](#)
 - _data_file_name
 - MeasureState, [84](#)
 - _data_mutex
 - Communication, [49](#)
 - _description
 - Command, [38](#)
 - _dev
 - SensorBME280, [101](#)
 - _error_state
 - StateContext, [113](#)
 - _event_flags
 - Buttons, [35](#)
 - CommandHandler, [43](#)
 - CommandState, [45](#)
 - Lora, [78](#)
 - MeasureState, [84](#)
 - _fat_file_system
 - SDCard, [96](#)
 - _file_name
 - Logger, [73](#)
 - _flags
 - Communication, [49](#)
 - _flash
 - ConfigCommand, [58](#)
 - InitState, [66](#)
 - _init_state
 - StateContext, [113](#)
 - _initialized
 - Logger, [73](#)
 - SDCard, [96](#)
 - _is_full
 - SDCard, [96](#)
 - _led_indicator
 - State, [110](#)
 - _lora
 - Communication, [49](#)
 - Protocol, [91](#)
 - _measure_state
 - StateContext, [113](#)
 - _measured_data
 - Communication, [49](#)
 - _measurement_ticker
 - MeasureState, [84](#)
 - _n_detect
 - SDCard, [96](#)

- `_name`
 - Command, 39
- `_new_file_ticker`
 - MeasureState, 84
- `_node_crc`
 - Protocol, 91
- `_node_id`
 - Protocol, 91
- `_protocol`
 - Communication, 49
- `_radio`
 - Lora, 78
- `_radio_callbacks`
 - Lora, 78
- `_rx_timeout`
 - Lora, 78
- `_sd_block_device`
 - SDCard, 96
- `_sd_card`
 - InitState, 66
 - Logger, 73
 - MeasureState, 85
- `_sd_card_mutex`
 - SDCard, 96
- `_serial`
 - SensorSPS30, 106
- `_spi`
 - SensorBME280, 101
- `_sps30`
 - InitState, 66
 - MeasureState, 85
 - SpsCommand, 108
- `_sps30_data_buffer`
 - MeasureState, 85
- `_sps30_info`
 - MeasureState, 85
- `_sps30_startup_timeout`
 - MeasureState, 85
- `_sps30_state`
 - MeasureState, 85
- `_status_led_green`
 - LedIndicator, 70
- `_status_led_red`
 - LedIndicator, 70
- `_status_led_timeout`
 - LedIndicator, 70
- `_status_update_ticker`
 - Communication, 49
- `_supply_enable`
 - SensorSPS30, 106
- `_thread`
 - Communication, 49
- `_usb_serial`
 - Command, 39
 - CommandHandler, 43
- `~BmeCommand`
 - BmeCommand, 31
- `~Command`
 - Command, 36
- `~CommandHandler`
 - CommandHandler, 40
- `~CommandState`
 - CommandState, 44
- `~ConfigCommand`
 - ConfigCommand, 57
- `~InfoCommand`
 - InfoCommand, 62
- `~InitState`
 - InitState, 64
- `~LedIndicator`
 - LedIndicator, 68
- `~Logger`
 - Logger, 71
- `~MeasureState`
 - MeasureState, 81
- `~SensorBME280`
 - SensorBME280, 98
- `~SensorSPS30`
 - SensorSPS30, 103
- `~SpsCommand`
 - SpsCommand, 107
- `~State`
 - State, 110
- `~StateContext`
 - StateContext, 112
- `~TimeCommand`
 - TimeCommand, 114
- `analogin_init_direct`
 - Battery.cpp, 133
- `app_version`
 - utils, 23
- `auth_reset`
 - Protocol, 88
- `authenticate`
 - Communication, 47
- `avg_data_t`
 - MeasureState, 80
- `base_ack_t`
 - Protocol, 92
- `base_auth_t`
 - Protocol, 92
- `bat_level_en`
 - Battery.cpp, 134
- `bat_level_input`
 - Battery.cpp, 134
- `Battery`, 28
 - read_voltage, 29
- `Battery.cpp`
 - analogin_init_direct, 133
 - bat_level_en, 134
 - bat_level_input, 134
- `BATTERY_THRESHOLD_LED_1`
 - LedIndicator.cpp, 135
- `BATTERY_THRESHOLD_LED_2`
 - LedIndicator.cpp, 135

BATTERY_THRESHOLD_LED_3
 LedIndicator.cpp, 135
bme280_cs
 SensorBME280::bme280_handle_t, 29
bme280_delay_us
 SensorBME280, 98
bme280_spi_init
 SensorBME280, 99
bme280_spi_read
 SensorBME280, 99
bme280_spi_write
 SensorBME280, 99
bme280_test
 InitState, 64
BME280_TEST_STARTUP_TIME_MS
 defines.h, 127
bme_command_description
 defines.h, 131
bme_command_name
 defines.h, 131
BmeCommand, 30
 _bme280, 32
 ~BmeCommand, 31
 BmeCommand, 31
 execute_command, 31
BOOL
 Config, 51
bool_v
 Config::config_value_t, 55
build
 version_number_s, 116
BUTTON_COMMAND_MIN_PRESS_TIME
 defines.h, 127
BUTTON_DEBOUNCE_TIME
 defines.h, 127
button_left_pressed
 Buttons, 33
button_left_released
 Buttons, 33
button_right_pressed
 Buttons, 33
button_right_released
 Buttons, 33
Buttons, 32
 _button_left, 34
 _button_left_timer, 34
 _button_right, 34
 _button_right_timer, 35
 _event_flags, 35
 button_left_pressed, 33
 button_left_released, 33
 button_right_pressed, 33
 button_right_released, 33
 Buttons, 33
 is_left_pressed, 34
 is_right_pressed, 34
cad_done_cb
 Lora, 75
calc_crc16
 utils, 23
calc_crc32
 utils, 24
calibrate
 TimeCommand, 115
CHANGELOG.md, 119
check_ack
 Protocol, 88
check_auth
 Protocol, 89
check_available_memory
 SDCard, 94
Color
 LedIndicator, 67
Command, 35
 _description, 38
 _name, 39
 _usb_serial, 39
 ~Command, 36
 Command, 36
 execute, 37
 execute_command, 37
 extract_parameters, 37
 get_description, 38
 get_name, 38
 split_string, 38
COMMAND_HANDLER_MAX_BUF_SIZE
 defines.h, 127
COMMAND_PRESSED
 State, 109
CommandHandler, 39
 _buffer, 42
 _commands, 43
 _event_flags, 43
 _usb_serial, 43
 ~CommandHandler, 40
 CommandHandler, 40
 connect, 40
 disconnect, 41
 find_command, 41
 handle_usb_serial_command, 41
 print_help, 41
 print_prompt, 41
 read_line_from_buffer, 42
 separate_command_name, 42
 serial_interrupt_handler, 42
CommandState, 43
 _command_handler, 45
 _event_flags, 45
 ~CommandState, 44
 CommandState, 44
 handle, 44
 run, 45
Communication, 46
 _data_mutex, 49
 _flags, 49
 _lora, 49

- [_measured_data](#), 49
 - [_protocol](#), 49
 - [_status_update_ticker](#), 49
 - [_thread](#), 49
 - [authenticate](#), 47
 - [Communication](#), 47
 - [MEASURED_DATA](#), 47
 - [PacketType](#), 46
 - [run](#), 47
 - [send_data](#), 47
 - [send_measured_data](#), 48
 - [send_status_data](#), 48
 - [start](#), 48
 - [STATUS_DATA](#), 47
 - [stop](#), 48
- [Communication.cpp](#)
 - [FLAG_SEND_MEASURED_DATA](#), 123
 - [FLAG_SEND_STATUS_DATA](#), 123
- [compose_log_line](#)
 - [Logger](#), 72
- [Config](#), 50
 - [BOOL](#), 51
 - [ConfigType](#), 51
 - [default_config](#), 54
 - [FLOAT](#), 51
 - [get](#), 51
 - [get_entry](#), 52
 - [global_config](#), 54
 - [read_config_from_flash](#), 52
 - [read_from_flash](#), 52
 - [reset_config](#), 53
 - [save_default_config](#), 53
 - [set_value](#), 53
 - [UINT32](#), 51
 - [write_config_to_flash](#), 53
 - [write_to_flash](#), 54
- [Config.h](#)
 - [NUM_CONFIG_ENTRIES](#), 125
- [Config::config_value_t](#), 55
 - [bool_v](#), 55
 - [float_v](#), 55
 - [uint32_v](#), 55
- [Config::entry_t](#), 58
 - [description](#), 58
 - [max](#), 59
 - [min](#), 59
 - [name](#), 59
 - [type](#), 59
 - [value](#), 59
- [config_command_description](#)
 - [defines.h](#), 131
- [config_command_name](#)
 - [defines.h](#), 131
- [ConfigCommand](#), 56
 - [_flash](#), 58
 - [~ConfigCommand](#), 57
 - [ConfigCommand](#), 56
 - [execute_command](#), 57
 - [handle_parameters](#), 57
 - [print_current_config](#), 58
- [ConfigType](#)
 - [Config](#), 51
- [connect](#)
 - [CommandHandler](#), 40
- [correct_data](#)
 - [SensorBME280](#), 100
 - [SensorSPS30](#), 103
- [crc_check](#)
 - [Protocol](#), 89
- [DATA_FILE_CREATION_PERIOD_SEC](#)
 - [defines.h](#), 127
- [default_config](#)
 - [Config](#), 54
- [defines.h](#)
 - [BME280_TEST_STARTUP_TIME_MS](#), 127
 - [bme_command_description](#), 131
 - [bme_command_name](#), 131
 - [BUTTON_COMMAND_MIN_PRESS_TIME](#), 127
 - [BUTTON_DEBOUNCE_TIME](#), 127
 - [COMMAND_HANDLER_MAX_BUF_SIZE](#), 127
 - [config_command_description](#), 131
 - [config_command_name](#), 131
 - [DATA_FILE_CREATION_PERIOD_SEC](#), 127
 - [FLAG_BUTTON_COMMAND](#), 128
 - [FLAG_BUTTON_MEASURE](#), 128
 - [FLAG_BUTTON_STATUS](#), 128
 - [FLAG_CREATE_NEW_FILE](#), 128
 - [FLAG_DO_MEASUREMENT](#), 128
 - [FLAG_LORA_RX_DONE](#), 128
 - [FLAG_LORA_RX_TIMEOUT](#), 128
 - [FLAG_LORA_TX_DONE](#), 129
 - [FLAG_RECEIVED_COMMAND](#), 129
 - [FLAG_SPS30_STARTUP_DONE](#), 129
 - [info_command_description](#), 131
 - [info_command_name](#), 132
 - [log_level_names](#), 132
 - [LogLevel](#), 130
 - [LORA_RESEND_WAIT_MS](#), 129
 - [LORA_RX_TIMEOUT_MS](#), 129
 - [LVL_DEBUG](#), 131
 - [LVL_ERROR](#), 131
 - [LVL_INFO](#), 131
 - [LVL_SUPPRESS](#), 131
 - [MAX_SEND_DATA_ATTEMPTS](#), 129
 - [NUM_LOG_LEVELS](#), 131
 - [sps_command_description](#), 132
 - [sps_command_name](#), 132
 - [STORAGE_ADDR_CONFIG](#), 130
 - [time_command_description](#), 132
 - [time_command_name](#), 133
 - [version](#), 133
 - [VERSION_BUILD](#), 130
 - [VERSION_MAJOR](#), 130
 - [VERSION_MINOR](#), 130
- [deinit](#)
 - [SDCard](#), 94

- description
 - Config::entry_t, 58
- disconnect
 - CommandHandler, 41
- display_battery_level_for
 - LedIndicator, 68
- display_status_for
 - LedIndicator, 68
- ErrorState, 59
 - _control_event_flags, 61
 - ErrorState, 60
 - handle, 60
- ErrorState.cpp
 - mbd_error_reboot_callback, 140
- execute
 - Command, 37
- execute_command
 - BmeCommand, 31
 - Command, 37
 - ConfigCommand, 57
 - InfoCommand, 62
 - SpsCommand, 107
 - TimeCommand, 115
- extract_parameters
 - Command, 37
- fhss_change_channel_cb
 - Lora, 75
- find_command
 - CommandHandler, 41
- FLAG_BUTTON_COMMAND
 - defines.h, 128
- FLAG_BUTTON_MEASURE
 - defines.h, 128
- FLAG_BUTTON_STATUS
 - defines.h, 128
- FLAG_CREATE_NEW_FILE
 - defines.h, 128
- FLAG_DO_MEASUREMENT
 - defines.h, 128
- FLAG_LORA_RX_DONE
 - defines.h, 128
- FLAG_LORA_RX_TIMEOUT
 - defines.h, 128
- FLAG_LORA_TX_DONE
 - defines.h, 129
- FLAG_RECEIVED_COMMAND
 - defines.h, 129
- FLAG_SEND_MEASURED_DATA
 - Communication.cpp, 123
- FLAG_SEND_STATUS_DATA
 - Communication.cpp, 123
- FLAG_SPS30_STARTUP_DONE
 - defines.h, 129
- FLASH_ERROR
 - State, 109
- FLOAT
 - Config, 51
- float_v
 - Config::config_value_t, 55
- format
 - SDCard, 94
- get
 - Config, 51
- get_description
 - Command, 38
- get_entry
 - Config, 52
- get_formatted_time_string
 - utils, 24
- get_hw_uuid
 - Protocol, 89
- get_name
 - Command, 38
- get_rand
 - utils, 24
- get_serial
 - SensorSPS30, 104
- get_sps30_info
 - MeasureState, 81
- get_sw_uuid
 - Protocol, 90
- global_config
 - Config, 54
- GREEN
 - LedIndicator, 67
- handle
 - CommandState, 44
 - ErrorState, 60
 - InitState, 64
 - MeasureState, 82
 - State, 110
- handle_parameters
 - ConfigCommand, 57
- handle_usb_serial_command
 - CommandHandler, 41
- hum
 - MeasureState::avg_data_s, 28
- info_command_description
 - defines.h, 131
- info_command_name
 - defines.h, 132
- InfoCommand, 61
 - ~InfoCommand, 62
 - execute_command, 62
 - InfoCommand, 61
- init
 - InitState, 64
 - Logger, 72
 - Lora, 76
 - SDCard, 95
 - SensorBME280, 100
- init_measurement_file
 - MeasureState, 82

- init_rtc
 - InitState, 65
- InitState, 62
 - _bme280, 65
 - _buttons, 65
 - _communication, 65
 - _flash, 66
 - _sd_card, 66
 - _sps30, 66
 - ~InitState, 64
 - bme280_test, 64
 - handle, 64
 - init, 64
 - init_rtc, 65
 - InitState, 63
 - sps30_test, 65
- InitState.cpp
 - TIMESTAMP_20200101, 141
- is_auth_done
 - Protocol, 90
- is_full
 - SDCard, 95
- is_inserted
 - SDCard, 95
- is_left_pressed
 - Buttons, 34
- is_number
 - utils, 25
- is_right_pressed
 - Buttons, 34
- LedIndicator, 66
 - _bat_led_1, 69
 - _bat_led_2, 70
 - _bat_led_3, 70
 - _bat_led_timeout, 70
 - _status_led_green, 70
 - _status_led_red, 70
 - _status_led_timeout, 70
 - ~LedIndicator, 68
 - Color, 67
 - display_battery_level_for, 68
 - display_status_for, 68
 - GREEN, 67
 - LedIndicator, 68
 - OFF, 67
 - ORANGE, 67
 - RED, 67
 - switch_battery_leds, 69
 - switch_status_leds, 69
 - turn_off_battery_leds, 69
 - turn_off_status_leds, 69
- LedIndicator.cpp
 - BATTERY_THRESHOLD_LED_1, 135
 - BATTERY_THRESHOLD_LED_2, 135
 - BATTERY_THRESHOLD_LED_3, 135
- LOG
 - Logger.h, 139
- log_internal
 - Logger, 73
- log_level_names
 - defines.h, 132
- Logger, 71
 - _file_name, 73
 - _initialized, 73
 - _sd_card, 73
 - ~Logger, 71
 - compose_log_line, 72
 - init, 72
 - log_internal, 73
 - Logger, 71, 72
 - operator=, 73
- Logger.h
 - __FILENAME__, 139
 - LOG, 139
- LogLevel
 - defines.h, 130
- Lora, 74
 - _event_flags, 78
 - _radio, 78
 - _radio_callbacks, 78
 - _rx_timeout, 78
 - cad_done_cb, 75
 - fhss_change_channel_cb, 75
 - init, 76
 - Lora, 75
 - receive, 76
 - rx_done_cb, 76
 - rx_error_cb, 76
 - rx_payload, 78
 - rx_payload_len, 78
 - rx_rssi, 78
 - rx_snr, 79
 - rx_timeout_cb, 76
 - send, 77
 - set_rx_config, 77
 - set_tx_config, 77
 - sleep, 77
 - tx_done_cb, 77
 - tx_timeout_cb, 77
- Lora.h
 - LORA_CFG_MODEM, 136
 - LORA_CFG_RX_BANDWIDTH_AFC, 136
 - LORA_CFG_RX_PAYLOAD_LEN, 136
 - LORA_CFG_TX_FDEV, 137
- LORA_CFG_MODEM
 - Lora.h, 136
- LORA_CFG_RX_BANDWIDTH_AFC
 - Lora.h, 136
- LORA_CFG_RX_PAYLOAD_LEN
 - Lora.h, 136
- LORA_CFG_TX_FDEV
 - Lora.h, 137
- LORA_ERROR
 - State, 109
- LORA_RESEND_WAIT_MS
 - defines.h, 129

LORA_RX_TIMEOUT_MS
 defines.h, 129
 LVL_DEBUG
 defines.h, 131
 LVL_ERROR
 defines.h, 131
 LVL_INFO
 defines.h, 131
 LVL_SUPPRESS
 defines.h, 131
 main
 main.cpp, 140
 main.cpp
 main, 140
 major
 version_number_s, 116
 max
 Config::entry_t, 59
 MAX_SEND_DATA_ATTEMPTS
 defines.h, 129
 mbed_error_reboot_callback
 ErrorState.cpp, 140
 MEASURED_DATA
 Communication, 47
 measured_data_t
 Protocol, 92
 MEASURED_DATA_TYPE_ID
 Protocol.cpp, 124
 measurement_t
 SensorBME280, 98
 SensorSPS30, 103
 MeasureState, 79
 _avg_data, 84
 _bme280, 84
 _communication, 84
 _data_file_name, 84
 _event_flags, 84
 _measurement_ticker, 84
 _new_file_ticker, 84
 _sd_card, 85
 _sps30, 85
 _sps30_data_buffer, 85
 _sps30_info, 85
 _sps30_startup_timeout, 85
 _sps30_state, 85
 ~MeasureState, 81
 avg_data_t, 80
 get_sps30_info, 81
 handle, 82
 init_measurement_file, 82
 MeasureState, 81
 OFF, 81
 run, 82
 RUNNING, 81
 send_measurement_data, 82
 set_create_new_file_flag, 83
 set_do_measurement_flag, 83
 set_sps30_startup_done_flag, 83
 SPS30State, 80
 STARTING, 81
 store_measurement_data, 83
 MeasureState::avg_data_s, 27
 hum, 28
 operator+/, 27
 operator/=: 27
 particle_size, 28
 pm2_5, 28
 press, 28
 temp, 28
 min
 Config::entry_t, 59
 minor
 version_number_s, 117
 MS_PER_DAY
 TimeCommand.cpp, 122
 name
 Config::entry_t, 59
 next_state
 StateContext, 112
 node_auth_t
 Protocol, 92
 node_measured_data_t
 Protocol, 92
 node_status_data_t
 Protocol, 92
 NUM_CONFIG_ENTRIES
 Config.h, 125
 NUM_LOG_LEVELS
 defines.h, 131
 OFF
 LedIndicator, 67
 MeasureState, 81
 operator+=
 MeasureState::avg_data_s, 27
 operator/=
 MeasureState::avg_data_s, 27
 operator=
 Logger, 73
 ORANGE
 LedIndicator, 67
 PacketType
 Communication, 46
 parse_datetime
 TimeCommand, 115
 particle_size
 MeasureState::avg_data_s, 28
 pm2_5
 MeasureState::avg_data_s, 28
 power_off
 SensorSPS30, 104
 power_on
 SensorSPS30, 104
 press
 MeasureState::avg_data_s, 28

- print_current_config
 - ConfigCommand, 58
- print_help
 - CommandHandler, 41
- print_prompt
 - CommandHandler, 41
- probe
 - SensorSPS30, 104
- Protocol, 86
 - __attribute__, 87, 88
 - _base_crc, 91
 - _client_auth_done, 91
 - _lora, 91
 - _node_crc, 91
 - _node_id, 91
 - auth_reset, 88
 - base_ack_t, 92
 - base_auth_t, 92
 - check_ack, 88
 - check_auth, 89
 - crc_check, 89
 - get_hw_uuid, 89
 - get_sw_uuid, 90
 - is_auth_done, 90
 - measured_data_t, 92
 - node_auth_t, 92
 - node_measured_data_t, 92
 - node_status_data_t, 92
 - Protocol, 87
 - read_received, 90
 - send_auth, 90
 - send_measured_data, 90
 - send_status_data, 91
 - status_data_t, 92
- Protocol.cpp
 - MEASURED_DATA_TYPE_ID, 124
 - SMARTMOTE_ACK, 124
 - SMARTMOTE_AUTH_ID, 124
 - SMARTMOTE_HW_ID, 124
 - STATUS_DATA_TYPE_ID, 124
- read_config_from_flash
 - Config, 52
- read_from_flash
 - Config, 52
- read_line_from_buffer
 - CommandHandler, 42
- read_measurement
 - SensorBME280, 100
 - SensorSPS30, 104
- read_received
 - Protocol, 90
- read_sensor_data
 - SensorBME280, 101
- read_version
 - SensorSPS30, 105
- read_voltage
 - Battery, 29
- README.md, 119
- receive
 - Lora, 76
- RED
 - LedIndicator, 67
- reset_config
 - Config, 53
- Result
 - State, 109
- REVERSE_2_BYTE
 - utils.h, 144
- run
 - CommandState, 45
 - Communication, 47
 - MeasureState, 82
 - StateContext, 112
- RUNNING
 - MeasureState, 81
- rx_done_cb
 - Lora, 76
- rx_error_cb
 - Lora, 76
- rx_payload
 - Lora, 78
- rx_payload_len
 - Lora, 78
- rx_rssi
 - Lora, 78
- rx_snr
 - Lora, 79
- rx_timeout_cb
 - Lora, 76
- save_default_config
 - Config, 53
- SD_CARD_ERROR
 - State, 109
- SDCard, 93
 - _fat_file_system, 96
 - _initialized, 96
 - _is_full, 96
 - _n_detect, 96
 - _sd_block_device, 96
 - _sd_card_mutex, 96
 - check_available_memory, 94
 - deinit, 94
 - format, 94
 - init, 95
 - is_full, 95
 - is_inserted, 95
 - SDCard, 94
 - write, 95
- send
 - Lora, 77
- send_auth
 - Protocol, 90
- send_data
 - Communication, 47
- send_measured_data
 - Communication, 48

- Protocol, 90
- send_measurement_data
 - MeasureState, 82
- send_status_data
 - Communication, 48
 - Protocol, 91
- SENSOR_ERROR
 - State, 109
- SensorBME280, 97
 - _bme280_cs, 101
 - _bme280_handle, 101
 - _dev, 101
 - _spi, 101
 - ~SensorBME280, 98
 - bme280_delay_us, 98
 - bme280_spi_init, 99
 - bme280_spi_read, 99
 - bme280_spi_write, 99
 - correct_data, 100
 - init, 100
 - measurement_t, 98
 - read_measurement, 100
 - read_sensor_data, 101
 - SensorBME280, 98
- SensorBME280::bme280_handle_t, 29
 - bme280_cs, 29
 - spi, 30
- SensorSPS30, 102
 - _serial, 106
 - _supply_enable, 106
 - ~SensorSPS30, 103
 - correct_data, 103
 - get_serial, 104
 - measurement_t, 103
 - power_off, 104
 - power_on, 104
 - probe, 104
 - read_measurement, 104
 - read_version, 105
 - SensorSPS30, 103
 - start_manual_fan_cleaning, 105
 - start_measurement, 105
 - stop_measurement, 105
- separate_command_name
 - CommandHandler, 42
- serial_interrupt_handler
 - CommandHandler, 42
- set_create_new_file_flag
 - MeasureState, 83
- set_do_measurement_flag
 - MeasureState, 83
- set_rx_config
 - Lora, 77
- set_sps30_startup_done_flag
 - MeasureState, 83
- set_tx_config
 - Lora, 77
- set_value
 - Config, 53
- sleep
 - Lora, 77
- SMARTMOTE_ACK
 - Protocol.cpp, 124
- SMARTMOTE_AUTH_ID
 - Protocol.cpp, 124
- SMARTMOTE_HW_ID
 - Protocol.cpp, 124
- spi
 - SensorBME280::bme280_handle_t, 30
- split_string
 - Command, 38
- sps30_test
 - InitState, 65
- SPS30State
 - MeasureState, 80
- sps_command_description
 - defines.h, 132
- sps_command_name
 - defines.h, 132
- SpsCommand, 106
 - _sps30, 108
 - ~SpsCommand, 107
 - execute_command, 107
 - SpsCommand, 107
- src/command/BmeCommand.cpp, 119
- src/command/BmeCommand.h, 119
- src/command/Command.cpp, 119
- src/command/Command.h, 120
- src/command/CommandHandler.cpp, 120
- src/command/CommandHandler.h, 120
- src/command/ConfigCommand.cpp, 120
- src/command/ConfigCommand.h, 121
- src/command/InfoCommand.cpp, 121
- src/command/InfoCommand.h, 121
- src/command/SpsCommand.cpp, 121
- src/command/SpsCommand.h, 121
- src/command/TimeCommand.cpp, 122
- src/command/TimeCommand.h, 122
- src/communication/Communication.cpp, 122
- src/communication/Communication.h, 123
- src/communication/Protocol.cpp, 123
- src/communication/Protocol.h, 124
- src/config/Config.cpp, 125
- src/config/Config.h, 125
- src/defs/defines.h, 125
- src/driver/Battery.cpp, 133
- src/driver/Battery.h, 134
- src/driver/Buttons.cpp, 134
- src/driver/Buttons.h, 134
- src/driver/LedIndicator.cpp, 135
- src/driver/LedIndicator.h, 135
- src/driver/Lora.cpp, 136
- src/driver/Lora.h, 136
- src/driver/SDCard.cpp, 137
- src/driver/SDCard.h, 137
- src/driver/SensorBME280.cpp, 137

- src/driver/SensorBME280.h, 137
- src/driver/SensorSPS30.cpp, 138
- src/driver/SensorSPS30.h, 138
- src/logging/Logger.cpp, 138
- src/logging/Logger.h, 138
- src/main.cpp, 139
- src/state/CommandState.cpp, 140
- src/state/CommandState.h, 140
- src/state/ErrorMessage.cpp, 140
- src/state/ErrorMessage.h, 141
- src/state/InitState.cpp, 141
- src/state/InitState.h, 141
- src/state/MeasureState.cpp, 142
- src/state/MeasureState.h, 142
- src/state/State.cpp, 142
- src/state/State.h, 142
- src/state/StateContext.cpp, 143
- src/state/StateContext.h, 143
- src/Utils/Utils.cpp, 143
- src/Utils/Utils.h, 144
- start
 - Communication, 48
- start_manual_fan_cleaning
 - SensorSPS30, 105
- start_measurement
 - SensorSPS30, 105
- STARTING
 - MeasureState, 81
- State, 108
 - _led_indicator, 110
 - ~State, 110
 - COMMAND_PRESSED, 109
 - FLASH_ERROR, 109
 - handle, 110
 - LORA_ERROR, 109
 - Result, 109
 - SD_CARD_ERROR, 109
 - SENSOR_ERROR, 109
 - State, 109
 - SUCCESS, 109
- StateContext, 111
 - _command_state, 112
 - _current_state, 113
 - _error_state, 113
 - _init_state, 113
 - _measure_state, 113
 - ~StateContext, 112
 - next_state, 112
 - run, 112
 - StateContext, 111
- STATUS_DATA
 - Communication, 47
- status_data_t
 - Protocol, 92
- STATUS_DATA_TYPE_ID
 - Protocol.cpp, 124
- stop
 - Communication, 48
- stop_measurement
 - SensorSPS30, 105
- STORAGE_ADDR_CONFIG
 - defines.h, 130
- store_measurement_data
 - MeasureState, 83
- string_to_bool
 - utils, 25
- SUCCESS
 - State, 109
- switch_battery_leds
 - LedIndicator, 69
- switch_status_leds
 - LedIndicator, 69
- temp
 - MeasureState::avg_data_s, 28
- time_command_description
 - defines.h, 132
- time_command_name
 - defines.h, 133
- TimeCommand, 113
 - _cal_time_sec, 116
 - ~TimeCommand, 114
 - calibrate, 115
 - execute_command, 115
 - parse_datetime, 115
 - TimeCommand, 114
 - wait_for_second_toggle, 116
- TimeCommand.cpp
 - MS_PER_DAY, 122
- TIMESTAMP_20200101
 - InitState.cpp, 141
- to_hex_string
 - utils, 25
- TRY
 - utils.h, 145
- turn_off_battery_leds
 - LedIndicator, 69
- turn_off_status_leds
 - LedIndicator, 69
- tx_done_cb
 - Lora, 77
- tx_timeout_cb
 - Lora, 77
- type
 - Config::entry_t, 59
- UINT32
 - Config, 51
- uint32_v
 - Config::config_value_t, 55
- utils, 23
 - app_version, 23
 - calc_crc16, 23
 - calc_crc32, 24
 - get_formatted_time_string, 24
 - get_rand, 24
 - is_number, 25

- string_to_bool, [25](#)
 - to_hex_string, [25](#)
- utils.h
 - REVERSE_2_BYTE, [144](#)
 - TRY, [145](#)
- value
 - Config::entry_t, [59](#)
- version
 - defines.h, [133](#)
- VERSION_BUILD
 - defines.h, [130](#)
- VERSION_MAJOR
 - defines.h, [130](#)
- VERSION_MINOR
 - defines.h, [130](#)
- version_number_s, [116](#)
 - build, [116](#)
 - major, [116](#)
 - minor, [117](#)
- wait_for_second_toggle
 - TimeCommand, [116](#)
- write
 - SDCard, [95](#)
- write_config_to_flash
 - Config, [53](#)
- write_to_flash
 - Config, [54](#)