

# KubiSat Firmware

Generated by Doxygen 1.13.2



---

<b>1 Clock Commands</b>	<b>1</b>
<b>2 Topic Index</b>	<b>3</b>
2.1 Topics . . . . .	3
<b>3 Hierarchical Index</b>	<b>5</b>
3.1 Class Hierarchy . . . . .	5
<b>4 Class Index</b>	<b>7</b>
4.1 Class List . . . . .	7
<b>5 File Index</b>	<b>9</b>
5.1 File List . . . . .	9
<b>6 Topic Documentation</b>	<b>11</b>
6.1 Clock Management Commands . . . . .	11
6.1.1 Detailed Description . . . . .	11
6.1.2 Function Documentation . . . . .	11
6.1.2.1 handle_time() . . . . .	11
6.1.2.2 handle_timezone_offset() . . . . .	12
6.1.2.3 handle_clock_sync_interval() . . . . .	13
6.1.2.4 handle_get_last_sync_time() . . . . .	14
6.1.3 Variable Documentation . . . . .	15
6.1.3.1 systemClock . . . . .	15
6.2 Command System . . . . .	15
6.2.1 Detailed Description . . . . .	15
6.2.2 Typedef Documentation . . . . .	15
6.2.2.1 CommandHandler . . . . .	15
6.2.2.2 CommandMap . . . . .	16
6.2.3 Function Documentation . . . . .	16
6.2.3.1 execute_command() . . . . .	16
6.2.4 Variable Documentation . . . . .	17
6.2.4.1 command_handlers . . . . .	17
6.3 Diagnostic Commands . . . . .	17
6.3.1 Detailed Description . . . . .	18
6.3.2 Function Documentation . . . . .	18
6.3.2.1 handle_get_commands_list() . . . . .	18
6.3.2.2 handle_get_build_version() . . . . .	18
6.3.2.3 handle_verbosity() . . . . .	19
6.3.2.4 handle_enter_bootloader_mode() . . . . .	21
6.3.3 Variable Documentation . . . . .	21
6.3.3.1 g_pending_bootloader_reset . . . . .	21
6.4 Event Commands . . . . .	21
6.4.1 Detailed Description . . . . .	22

---

6.4.2 Function Documentation . . . . .	22
6.4.2.1 handle_get_last_events() . . . . .	22
6.4.2.2 handle_get_event_count() . . . . .	23
6.5 GPS Commands . . . . .	24
6.5.1 Detailed Description . . . . .	24
6.5.2 Function Documentation . . . . .	24
6.5.2.1 handle_gps_power_status() . . . . .	24
6.5.2.2 handle_enable_gps_uart_passthrough() . . . . .	25
6.5.2.3 handle_get_rmc_data() . . . . .	26
6.5.2.4 handle_get_gga_data() . . . . .	27
6.6 Power Commands . . . . .	28
6.6.1 Detailed Description . . . . .	28
6.6.2 Function Documentation . . . . .	28
6.6.2.1 handle_get_power_manager_ids() . . . . .	28
6.6.2.2 handle_get_voltage_battery() . . . . .	29
6.6.2.3 handle_get_voltage_5v() . . . . .	30
6.6.2.4 handle_get_current_charge_usb() . . . . .	31
6.6.2.5 handle_get_current_charge_solar() . . . . .	32
6.6.2.6 handle_get_current_charge_total() . . . . .	32
6.6.2.7 handle_get_current_draw() . . . . .	33
6.7 Sensor Commands . . . . .	34
6.7.1 Detailed Description . . . . .	34
6.7.2 Function Documentation . . . . .	34
6.7.2.1 handle_get_sensor_data() . . . . .	34
6.7.2.2 handle_sensor_config() . . . . .	35
6.7.2.3 handle_get_sensor_list() . . . . .	36
6.8 Storage Commands . . . . .	37
6.8.1 Detailed Description . . . . .	38
6.8.2 Function Documentation . . . . .	38
6.8.2.1 handle_list_files() . . . . .	38
6.8.2.2 handle_mount() . . . . .	38
6.9 Frame Handling . . . . .	39
6.9.1 Detailed Description . . . . .	40
6.9.2 Function Documentation . . . . .	40
6.9.2.1 frame_encode() . . . . .	40
6.9.2.2 frame_decode() . . . . .	41
6.9.2.3 frame_process() . . . . .	42
6.9.2.4 frame_build() . . . . .	43
6.10 Event Manager . . . . .	44
6.10.1 Detailed Description . . . . .	46
6.10.2 Enumeration Type Documentation . . . . .	46
6.10.2.1 EventGroup . . . . .	46

---

6.10.2.2 SystemEvent . . . . .	46
6.10.2.3 PowerEvent . . . . .	47
6.10.2.4 CommsEvent . . . . .	47
6.10.2.5 GPSEvent . . . . .	47
6.10.2.6 ClockEvent . . . . .	48
6.10.3 Function Documentation . . . . .	48
6.10.3.1 __attribute__() . . . . .	48
6.10.3.2 log_event() . . . . .	48
6.10.3.3 get_event() . . . . .	49
6.10.4 Variable Documentation . . . . .	50
6.10.4.1 eventLogId . . . . .	50
6.10.4.2 systemClock . . . . .	50
6.10.4.3 eventManager [1/2] . . . . .	50
6.10.4.4 __attribute__ . . . . .	50
6.10.4.5 eventManager [2/2] . . . . .	50
6.11 INA3221 Power Monitor . . . . .	51
6.11.1 Detailed Description . . . . .	51
6.11.2 Configuration Functions . . . . .	51
6.11.2.1 Detailed Description . . . . .	52
6.11.2.2 Function Documentation . . . . .	52
6.11.3 Measurement Functions . . . . .	61
6.11.3.1 Detailed Description . . . . .	61
6.11.3.2 Function Documentation . . . . .	61
6.11.4 Alert Functions . . . . .	63
6.11.4.1 Detailed Description . . . . .	64
6.11.4.2 Function Documentation . . . . .	64
6.12 Telemetry Manager . . . . .	69
6.12.1 Detailed Description . . . . .	70
6.12.2 Function Documentation . . . . .	70
6.12.2.1 telemetry_init() . . . . .	70
6.12.2.2 collect_telemetry() . . . . .	71
6.12.2.3 flush_telemetry() . . . . .	71
6.12.2.4 is_telemetry_collection_time() . . . . .	72
6.12.2.5 is_telemetry_flush_time() . . . . .	73
6.12.2.6 get_telemetry_sample_interval() . . . . .	74
6.12.2.7 set_telemetry_sample_interval() . . . . .	74
6.12.2.8 get_telemetry_flush_threshold() . . . . .	74
6.12.2.9 set_telemetry_flush_threshold() . . . . .	74
<b>7 Class Documentation</b> . . . . .	<b>75</b>
7.1 BH1750 Class Reference . . . . .	75
7.1.1 Detailed Description . . . . .	75

---

7.1.2 Member Enumeration Documentation . . . . .	75
7.1.2.1 Mode . . . . .	75
7.1.3 Constructor & Destructor Documentation . . . . .	76
7.1.3.1 BH1750() . . . . .	76
7.1.4 Member Function Documentation . . . . .	76
7.1.4.1 begin() . . . . .	76
7.1.4.2 configure() . . . . .	77
7.1.4.3 get_light_level() . . . . .	77
7.1.4.4 write8() . . . . .	77
7.1.5 Member Data Documentation . . . . .	78
7.1.5.1 _i2c_addr . . . . .	78
2.2 BH1750Wrapper Class Reference . . . . .	78
7.2.1 Detailed Description . . . . .	79
7.2.2 Constructor & Destructor Documentation . . . . .	79
7.2.2.1 BH1750Wrapper() . . . . .	79
7.2.3 Member Function Documentation . . . . .	79
7.2.3.1 get_i2c_addr() . . . . .	79
7.2.3.2 init() . . . . .	79
7.2.3.3 read_data() . . . . .	80
7.2.3.4 is_initialized() . . . . .	80
7.2.3.5 get_type() . . . . .	80
7.2.3.6 configure() . . . . .	80
7.2.3.7 get_address() . . . . .	80
7.2.4 Member Data Documentation . . . . .	80
7.2.4.1 sensor_ . . . . .	80
7.2.4.2 initialized_ . . . . .	81
2.3 BME280 Class Reference . . . . .	81
7.3.1 Detailed Description . . . . .	82
7.3.2 Constructor & Destructor Documentation . . . . .	83
7.3.2.1 BME280() . . . . .	83
7.3.3 Member Function Documentation . . . . .	83
7.3.3.1 init() . . . . .	83
7.3.3.2 reset() . . . . .	83
7.3.3.3 read_raw_all() . . . . .	83
7.3.3.4 convert_temperature() . . . . .	84
7.3.3.5 convert_pressure() . . . . .	84
7.3.3.6 convert_humidity() . . . . .	84
7.3.3.7 configure_sensor() . . . . .	84
7.3.3.8 get_calibration_parameters() . . . . .	84
7.3.4 Member Data Documentation . . . . .	85
7.3.4.1 ADDR_SDO_LOW . . . . .	85
7.3.4.2 ADDR_SDO_HIGH . . . . .	85

---

7.3.4.3 i2c_port . . . . .	85
7.3.4.4 device_addr . . . . .	85
7.3.4.5 calib_params . . . . .	85
7.3.4.6 initialized_ . . . . .	85
7.3.4.7 t_fine . . . . .	85
7.3.4.8 REG_CONFIG . . . . .	86
7.3.4.9 REG_CTRL_MEAS . . . . .	86
7.3.4.10 REG_CTRL_HUM . . . . .	86
7.3.4.11 REG_RESET . . . . .	86
7.3.4.12 REG_PRESSURE_MSB . . . . .	86
7.3.4.13 REG_TEMPERATURE_MSB . . . . .	86
7.3.4.14 REG_HUMIDITY_MSB . . . . .	86
7.3.4.15 REG_DIG_T1_LSB . . . . .	86
7.3.4.16 REG_DIG_T1_MSB . . . . .	87
7.3.4.17 REG_DIG_T2_LSB . . . . .	87
7.3.4.18 REG_DIG_T2_MSB . . . . .	87
7.3.4.19 REG_DIG_T3_LSB . . . . .	87
7.3.4.20 REG_DIG_T3_MSB . . . . .	87
7.3.4.21 REG_DIG_P1_LSB . . . . .	87
7.3.4.22 REG_DIG_P1_MSB . . . . .	87
7.3.4.23 REG_DIG_P2_LSB . . . . .	87
7.3.4.24 REG_DIG_P2_MSB . . . . .	88
7.3.4.25 REG_DIG_P3_LSB . . . . .	88
7.3.4.26 REG_DIG_P3_MSB . . . . .	88
7.3.4.27 REG_DIG_P4_LSB . . . . .	88
7.3.4.28 REG_DIG_P4_MSB . . . . .	88
7.3.4.29 REG_DIG_P5_LSB . . . . .	88
7.3.4.30 REG_DIG_P5_MSB . . . . .	88
7.3.4.31 REG_DIG_P6_LSB . . . . .	88
7.3.4.32 REG_DIG_P6_MSB . . . . .	89
7.3.4.33 REG_DIG_P7_LSB . . . . .	89
7.3.4.34 REG_DIG_P7_MSB . . . . .	89
7.3.4.35 REG_DIG_P8_LSB . . . . .	89
7.3.4.36 REG_DIG_P8_MSB . . . . .	89
7.3.4.37 REG_DIG_P9_LSB . . . . .	89
7.3.4.38 REG_DIG_P9_MSB . . . . .	89
7.3.4.39 REG_DIG_H1 . . . . .	89
7.3.4.40 REG_DIG_H2 . . . . .	90
7.3.4.41 REG_DIG_H3 . . . . .	90
7.3.4.42 REG_DIG_H4 . . . . .	90
7.3.4.43 REG_DIG_H5 . . . . .	90
7.3.4.44 REG_DIG_H6 . . . . .	90

---

7.3.4.45 NUM_CALIB_PARAMS . . . . .	90
7.4 BME280CalibParam Struct Reference . . . . .	90
7.4.1 Detailed Description . . . . .	91
7.4.2 Member Data Documentation . . . . .	91
7.4.2.1 dig_t1 . . . . .	91
7.4.2.2 dig_t2 . . . . .	91
7.4.2.3 dig_t3 . . . . .	91
7.4.2.4 dig_p1 . . . . .	92
7.4.2.5 dig_p2 . . . . .	92
7.4.2.6 dig_p3 . . . . .	92
7.4.2.7 dig_p4 . . . . .	92
7.4.2.8 dig_p5 . . . . .	92
7.4.2.9 dig_p6 . . . . .	92
7.4.2.10 dig_p7 . . . . .	92
7.4.2.11 dig_p8 . . . . .	92
7.4.2.12 dig_p9 . . . . .	93
7.4.2.13 dig_h1 . . . . .	93
7.4.2.14 dig_h2 . . . . .	93
7.4.2.15 dig_h3 . . . . .	93
7.4.2.16 dig_h4 . . . . .	93
7.4.2.17 dig_h5 . . . . .	93
7.4.2.18 dig_h6 . . . . .	93
7.5 BME280Wrapper Class Reference . . . . .	94
7.5.1 Detailed Description . . . . .	95
7.5.2 Constructor & Destructor Documentation . . . . .	95
7.5.2.1 BME280Wrapper() . . . . .	95
7.5.3 Member Function Documentation . . . . .	95
7.5.3.1 init() . . . . .	95
7.5.3.2 read_data() . . . . .	95
7.5.3.3 is_initialized() . . . . .	95
7.5.3.4 get_type() . . . . .	96
7.5.3.5 configure() . . . . .	96
7.5.3.6 get_address() . . . . .	96
7.5.4 Member Data Documentation . . . . .	96
7.5.4.1 sensor_ . . . . .	96
7.5.4.2 initialized_ . . . . .	96
7.6 INA3221::conf_reg_t Struct Reference . . . . .	96
7.6.1 Detailed Description . . . . .	97
7.6.2 Member Data Documentation . . . . .	97
7.6.2.1 mode_shunt_en . . . . .	97
7.6.2.2 mode_bus_en . . . . .	97
7.6.2.3 mode_continious_en . . . . .	97

---

7.6.2.4 shunt_conv_time . . . . .	97
7.6.2.5 bus_conv_time . . . . .	98
7.6.2.6 avg_mode . . . . .	98
7.6.2.7 ch3_en . . . . .	98
7.6.2.8 ch2_en . . . . .	98
7.6.2.9 ch1_en . . . . .	98
7.6.2.10 reset . . . . .	98
7.7 DS3231 Class Reference . . . . .	98
7.7.1 Detailed Description . . . . .	100
7.7.2 Constructor & Destructor Documentation . . . . .	100
7.7.2.1 DS3231() . . . . .	100
7.7.3 Member Function Documentation . . . . .	100
7.7.3.1 set_time() . . . . .	100
7.7.3.2 get_time() . . . . .	101
7.7.3.3 read_temperature() . . . . .	103
7.7.3.4 set_unix_time() . . . . .	104
7.7.3.5 get_unix_time() . . . . .	105
7.7.3.6 clock_enable() . . . . .	106
7.7.3.7 get_timezone_offset() . . . . .	107
7.7.3.8 set_timezone_offset() . . . . .	107
7.7.3.9 get_clock_sync_interval() . . . . .	108
7.7.3.10 set_clock_sync_interval() . . . . .	108
7.7.3.11 get_last_sync_time() . . . . .	109
7.7.3.12 update_last_sync_time() . . . . .	110
7.7.3.13 get_local_time() . . . . .	110
7.7.3.14 is_sync_needed() . . . . .	111
7.7.3.15 sync_clock_with_gps() . . . . .	111
7.7.3.16 i2c_read_reg() . . . . .	112
7.7.3.17 i2c_write_reg() . . . . .	113
7.7.3.18 bin_to_bcd() . . . . .	115
7.7.3.19 bcd_to_bin() . . . . .	115
7.7.4 Member Data Documentation . . . . .	116
7.7.4.1 i2c . . . . .	116
7.7.4.2 ds3231_addr . . . . .	116
7.7.4.3 clock_mutex_ . . . . .	117
7.7.4.4 timezone_offset_minutes_ . . . . .	117
7.7.4.5 sync_interval_minutes_ . . . . .	117
7.7.4.6 last_sync_time_ . . . . .	117
7.8 ds3231_data_t Struct Reference . . . . .	117
7.8.1 Detailed Description . . . . .	118
7.8.2 Member Data Documentation . . . . .	118
7.8.2.1 seconds . . . . .	118

7.8.2.2 minutes . . . . .	118
7.8.2.3 hours . . . . .	118
7.8.2.4 day . . . . .	119
7.8.2.5 date . . . . .	119
7.8.2.6 month . . . . .	119
7.8.2.7 year . . . . .	119
7.8.2.8 century . . . . .	119
7.9 EventEmitter Class Reference . . . . .	119
7.9.1 Detailed Description . . . . .	120
7.9.2 Member Function Documentation . . . . .	120
7.9.2.1 emit() . . . . .	120
7.10 EventLog Class Reference . . . . .	121
7.10.1 Detailed Description . . . . .	122
7.10.2 Member Function Documentation . . . . .	122
7.10.2.1 to_string() . . . . .	122
7.10.3 Member Data Documentation . . . . .	122
7.10.3.1 id . . . . .	122
7.10.3.2 timestamp . . . . .	122
7.10.3.3 group . . . . .	123
7.10.3.4 event . . . . .	123
7.11 EventManager Class Reference . . . . .	123
7.11.1 Detailed Description . . . . .	124
7.11.2 Constructor & Destructor Documentation . . . . .	124
7.11.2.1 EventManager() . . . . .	124
7.11.2.2 ~EventManager() . . . . .	125
7.11.3 Member Function Documentation . . . . .	125
7.11.3.1 init() . . . . .	125
7.11.3.2 get_event_count() . . . . .	125
7.11.3.3 save_to_storage() . . . . .	126
7.11.3.4 load_from_storage() . . . . .	126
7.11.4 Member Data Documentation . . . . .	126
7.11.4.1 events . . . . .	126
7.11.4.2 eventCount . . . . .	127
7.11.4.3 writeIndex . . . . .	127
7.11.4.4 eventMutex . . . . .	127
7.11.4.5 nextEventId . . . . .	127
7.11.4.6 eventsSinceFlush . . . . .	127
7.12 EventManagerImpl Class Reference . . . . .	128
7.12.1 Detailed Description . . . . .	129
7.12.2 Constructor & Destructor Documentation . . . . .	130
7.12.2.1 EventManagerImpl() . . . . .	130
7.12.3 Member Function Documentation . . . . .	130

---

7.12.3.1 save_to_storage() . . . . .	130
7.12.3.2 load_from_storage() . . . . .	131
7.13 FileHandle Struct Reference . . . . .	131
7.13.1 Detailed Description . . . . .	131
7.13.2 Member Data Documentation . . . . .	131
7.13.2.1 fd . . . . .	131
7.13.2.2 is_open . . . . .	131
7.14 Frame Struct Reference . . . . .	132
7.14.1 Detailed Description . . . . .	132
7.14.2 Member Data Documentation . . . . .	133
7.14.2.1 header . . . . .	133
7.14.2.2 direction . . . . .	133
7.14.2.3 operationType . . . . .	133
7.14.2.4 group . . . . .	133
7.14.2.5 command . . . . .	134
7.14.2.6 value . . . . .	134
7.14.2.7 unit . . . . .	134
7.14.2.8 footer . . . . .	134
7.15 HMC5883L Class Reference . . . . .	134
7.15.1 Detailed Description . . . . .	135
7.15.2 Constructor & Destructor Documentation . . . . .	135
7.15.2.1 HMC5883L() . . . . .	135
7.15.3 Member Function Documentation . . . . .	135
7.15.3.1 init() . . . . .	135
7.15.3.2 read() . . . . .	136
7.15.3.3 write_register() . . . . .	136
7.15.3.4 read_register() . . . . .	136
7.15.4 Member Data Documentation . . . . .	137
7.15.4.1 i2c . . . . .	137
7.15.4.2 address . . . . .	137
7.16 HMC5883LWrapper Class Reference . . . . .	137
7.16.1 Detailed Description . . . . .	138
7.16.2 Constructor & Destructor Documentation . . . . .	138
7.16.2.1 HMC5883LWrapper() . . . . .	138
7.16.3 Member Function Documentation . . . . .	139
7.16.3.1 init() . . . . .	139
7.16.3.2 read_data() . . . . .	139
7.16.3.3 is_initialized() . . . . .	139
7.16.3.4 get_type() . . . . .	139
7.16.3.5 configure() . . . . .	139
7.16.3.6 get_address() . . . . .	139
7.16.4 Member Data Documentation . . . . .	140

---

7.16.4.1 <code>sensor_</code>	140
7.16.4.2 <code>initialized_</code>	140
7.17 INA3221 Class Reference	140
7.17.1 Detailed Description	142
7.17.2 Member Function Documentation	142
7.17.2.1 <code>_read()</code>	142
7.17.2.2 <code>_write()</code>	144
7.17.2.3 <code>get_current()</code>	145
7.17.3 Member Data Documentation	146
7.17.3.1 <code>_i2c</code>	146
7.17.3.2 <code>_i2c_addr</code>	146
7.17.3.3 <code>_shuntRes</code>	146
7.17.3.4 <code>_filterRes</code>	146
7.17.3.5 <code>_masken_reg</code>	146
7.18 ISensor Class Reference	146
7.18.1 Detailed Description	147
7.18.2 Constructor & Destructor Documentation	147
7.18.2.1 <code>~ISensor()</code>	147
7.18.3 Member Function Documentation	147
7.18.3.1 <code>init()</code>	147
7.18.3.2 <code>read_data()</code>	147
7.18.3.3 <code>is_initialized()</code>	147
7.18.3.4 <code>get_type()</code>	147
7.18.3.5 <code>configure()</code>	148
7.18.3.6 <code>get_address()</code>	148
7.19 INA3221::masken_reg_t Struct Reference	148
7.19.1 Detailed Description	148
7.19.2 Member Data Documentation	149
7.19.2.1 <code>conv_ready</code>	149
7.19.2.2 <code>timing_ctrl_alert</code>	149
7.19.2.3 <code>pwr_valid_alert</code>	149
7.19.2.4 <code>warn_alert_ch3</code>	149
7.19.2.5 <code>warn_alert_ch2</code>	149
7.19.2.6 <code>warn_alert_ch1</code>	149
7.19.2.7 <code>shunt_sum_alert</code>	149
7.19.2.8 <code>crit_alert_ch3</code>	150
7.19.2.9 <code>crit_alert_ch2</code>	150
7.19.2.10 <code>crit_alert_ch1</code>	150
7.19.2.11 <code>crit_alert_latch_en</code>	150
7.19.2.12 <code>warn_alert_latch_en</code>	150
7.19.2.13 <code>shunt_sum_en_ch3</code>	150
7.19.2.14 <code>shunt_sum_en_ch2</code>	150

---

7.19.2.15 shunt_sum_en_ch1 . . . . .	150
7.19.2.16 reserved . . . . .	151
7.20 MPU6050Wrapper Class Reference . . . . .	151
7.20.1 Detailed Description . . . . .	152
7.20.2 Constructor & Destructor Documentation . . . . .	152
7.20.2.1 MPU6050Wrapper() . . . . .	152
7.20.3 Member Function Documentation . . . . .	152
7.20.3.1 init() . . . . .	152
7.20.3.2 read_data() . . . . .	152
7.20.3.3 is_initialized() . . . . .	152
7.20.3.4 get_type() . . . . .	152
7.20.3.5 configure() . . . . .	153
7.20.4 Member Data Documentation . . . . .	153
7.20.4.1 sensor_ . . . . .	153
7.20.4.2 initialized_ . . . . .	153
7.21 NMEAData Class Reference . . . . .	153
7.21.1 Detailed Description . . . . .	154
7.21.2 Constructor & Destructor Documentation . . . . .	154
7.21.2.1 NMEAData() . . . . .	154
7.21.3 Member Function Documentation . . . . .	154
7.21.3.1 update_rmc_tokens() . . . . .	154
7.21.3.2 update_gga_tokens() . . . . .	154
7.21.3.3 get_rmc_tokens() . . . . .	154
7.21.3.4 get_gga_tokens() . . . . .	154
7.21.3.5 has_valid_time() . . . . .	155
7.21.3.6 get_unix_time() . . . . .	155
7.21.4 Member Data Documentation . . . . .	155
7.21.4.1 rmc_tokens_ . . . . .	155
7.21.4.2 gga_tokens_ . . . . .	155
7.21.4.3 rmc_mutex_ . . . . .	156
7.21.4.4 gga_mutex_ . . . . .	156
7.22 PowerManager Class Reference . . . . .	156
7.22.1 Detailed Description . . . . .	157
7.22.2 Constructor & Destructor Documentation . . . . .	157
7.22.2.1 PowerManager() . . . . .	157
7.22.3 Member Function Documentation . . . . .	158
7.22.3.1 initialize() . . . . .	158
7.22.3.2 read_device_ids() . . . . .	158
7.22.3.3 get_current_charge_solar() . . . . .	158
7.22.3.4 get_current_charge_usb() . . . . .	158
7.22.3.5 get_current_charge_total() . . . . .	159
7.22.3.6 get_current_draw() . . . . .	159

7.22.3.7 get_voltage_battery()	159
7.22.3.8 get_voltage_5v()	159
7.22.3.9 configure()	159
7.22.3.10 is_charging_solar()	159
7.22.3.11 is_charging_usb()	160
7.22.3.12 check_power_alerts()	160
7.22.4 Member Data Documentation	160
7.22.4.1 SOLAR_CURRENT_THRESHOLD	160
7.22.4.2 USB_CURRENT_THRESHOLD	160
7.22.4.3 VOLTAGE_LOW_THRESHOLD	161
7.22.4.4 VOLTAGE_OVERCHARGE_THRESHOLD	161
7.22.4.5 FALL_RATE_THRESHOLD	161
7.22.4.6 FALLING_TREND_REQUIRED	161
7.22.4.7 ina3221_	161
7.22.4.8 initialized_	161
7.22.4.9 powerman_mutex_	161
7.22.4.10 charging_solar_active_	161
7.22.4.11 charging_usb_active_	162
7.23 SensorWrapper Class Reference	162
7.23.1 Detailed Description	163
7.23.2 Constructor & Destructor Documentation	163
7.23.2.1 SensorWrapper()	163
7.23.3 Member Function Documentation	163
7.23.3.1 get_instance()	163
7.23.3.2 sensor_init()	164
7.23.3.3 sensor_configure()	164
7.23.3.4 sensor_read_data()	165
7.23.3.5 get_sensor()	166
7.23.3.6 scan_connected_sensors()	166
7.23.3.7 get_available_sensors()	166
7.23.4 Member Data Documentation	167
7.23.4.1 sensors	167
7.24 TelemetryRecord Struct Reference	167
7.24.1 Detailed Description	168
7.24.2 Member Function Documentation	168
7.24.2.1 to_csv()	168
7.24.3 Member Data Documentation	168
7.24.3.1 timestamp	168
7.24.3.2 battery_voltage	168
7.24.3.3 system_voltage	168
7.24.3.4 charge_current_usb	168
7.24.3.5 charge_current_solar	169

---

7.24.3.6 discharge_current . . . . .	169
7.24.3.7 temperature . . . . .	169
7.24.3.8 pressure . . . . .	169
7.24.3.9 humidity . . . . .	169
7.24.3.10 light_level . . . . .	169
7.24.3.11 time . . . . .	170
7.24.3.12 latitude . . . . .	170
7.24.3.13 lat_dir . . . . .	170
7.24.3.14 longitude . . . . .	170
7.24.3.15 lon_dir . . . . .	170
7.24.3.16 speed . . . . .	170
7.24.3.17 course . . . . .	171
7.24.3.18 date . . . . .	171
7.24.3.19 fix_quality . . . . .	171
7.24.3.20 satellites . . . . .	171
7.24.3.21 altitude . . . . .	171
<b>8 File Documentation</b>	<b>173</b>
8.1 build_number.h File Reference . . . . .	173
8.1.1 Macro Definition Documentation . . . . .	173
8.1.1.1 BUILD_NUMBER . . . . .	173
8.2 build_number.h . . . . .	173
8.3 includes.h File Reference . . . . .	174
8.4 includes.h . . . . .	175
8.5 lib/clock/DS3231.cpp File Reference . . . . .	175
8.6 DS3231.cpp . . . . .	175
8.7 lib/clock/DS3231.h File Reference . . . . .	180
8.7.1 Macro Definition Documentation . . . . .	181
8.7.1.1 DS3231_DEVICE_ADRESS . . . . .	181
8.7.1.2 DS3231_SECONDS_REG . . . . .	181
8.7.1.3 DS3231_MINUTES_REG . . . . .	182
8.7.1.4 DS3231_HOURS_REG . . . . .	182
8.7.1.5 DS3231_DAY_REG . . . . .	182
8.7.1.6 DS3231_DATE_REG . . . . .	182
8.7.1.7 DS3231_MONTH_REG . . . . .	182
8.7.1.8 DS3231_YEAR_REG . . . . .	182
8.7.1.9 DS3231_CONTROL_REG . . . . .	183
8.7.1.10 DS3231_CONTROL_STATUS_REG . . . . .	183
8.7.1.11 DS3231_TEMPERATURE_MSB_REG . . . . .	183
8.7.1.12 DS3231_TEMPERATURE_LSB_REG . . . . .	183
8.7.2 Enumeration Type Documentation . . . . .	183
8.7.2.1 days_of_week . . . . .	183

---

8.8 DS3231.h . . . . .	184
8.9 lib/comms/commands/clock_commands.cpp File Reference . . . . .	185
8.9.1 Macro Definition Documentation . . . . .	186
8.9.1.1 CLOCK_GROUP . . . . .	186
8.9.1.2 TIME . . . . .	186
8.9.1.3 TIMEZONE_OFFSET . . . . .	186
8.9.1.4 CLOCK_SYNC_INTERVAL . . . . .	186
8.9.1.5 LAST_SYNC_TIME . . . . .	186
8.10 clock_commands.cpp . . . . .	187
8.11 lib/comms/commands/commands.cpp File Reference . . . . .	189
8.12 commands.cpp . . . . .	190
8.13 lib/comms/commands/commands.h File Reference . . . . .	191
8.14 commands.h . . . . .	193
8.15 lib/comms/commands/diagnostic_commands.cpp File Reference . . . . .	193
8.16 diagnostic_commands.cpp . . . . .	194
8.17 lib/comms/commands/event_commands.cpp File Reference . . . . .	196
8.18 event_commands.cpp . . . . .	196
8.19 lib/comms/commands/gps_commands.cpp File Reference . . . . .	197
8.19.1 Macro Definition Documentation . . . . .	198
8.19.1.1 GPS_GROUP . . . . .	198
8.19.1.2 POWER_STATUS_COMMAND . . . . .	198
8.19.1.3 PASSTHROUGH_COMMAND . . . . .	198
8.19.1.4 RMC_DATA_COMMAND . . . . .	198
8.19.1.5 GGA_DATA_COMMAND . . . . .	198
8.20 gps_commands.cpp . . . . .	199
8.21 lib/comms/commands/power_commands.cpp File Reference . . . . .	201
8.21.1 Macro Definition Documentation . . . . .	202
8.21.1.1 POWER_GROUP . . . . .	202
8.21.1.2 POWER_MANAGER_IDS . . . . .	202
8.21.1.3 VOLTAGE_BATTERY . . . . .	202
8.21.1.4 VOLTAGE_MAIN . . . . .	202
8.21.1.5 CHARGE_USB . . . . .	203
8.21.1.6 CHARGE_SOLAR . . . . .	203
8.21.1.7 CHARGE_TOTAL . . . . .	203
8.21.1.8 DRAW_TOTAL . . . . .	203
8.22 power_commands.cpp . . . . .	203
8.23 lib/comms/commands/sensor_commands.cpp File Reference . . . . .	205
8.23.1 Macro Definition Documentation . . . . .	206
8.23.1.1 SENSOR_GROUP . . . . .	206
8.23.1.2 SENSOR_READ . . . . .	206
8.23.1.3 SENSOR_CONFIGURE . . . . .	206
8.24 sensor_commands.cpp . . . . .	207

---

8.25 lib/comms/commands/storage_commands.cpp File Reference . . . . .	210
8.25.1 Macro Definition Documentation . . . . .	211
8.25.1.1 STORAGE_GROUP . . . . .	211
8.25.1.2 LIST_FILES_COMMAND . . . . .	211
8.25.1.3 MOUNT_COMMAND . . . . .	211
8.26 storage_commands.cpp . . . . .	212
8.27 lib/comms/communication.cpp File Reference . . . . .	213
8.27.1 Function Documentation . . . . .	214
8.27.1.1 initialize_radio() . . . . .	214
8.27.2 Variable Documentation . . . . .	215
8.27.2.1 outgoing . . . . .	215
8.27.2.2 msgCount . . . . .	215
8.27.2.3 lastSendTime . . . . .	215
8.27.2.4 lastReceiveTime . . . . .	215
8.27.2.5 lastPrintTime . . . . .	215
8.27.2.6 interval . . . . .	215
8.28 communication.cpp . . . . .	216
8.29 lib/comms/communication.h File Reference . . . . .	216
8.29.1 Function Documentation . . . . .	217
8.29.1.1 initialize_radio() . . . . .	217
8.29.1.2 on_receive() . . . . .	218
8.29.1.3 handle_uart_input() . . . . .	219
8.29.1.4 send_message() . . . . .	219
8.29.1.5 send_frame_uart() . . . . .	219
8.29.1.6 send_frame_lora() . . . . .	220
8.29.1.7 split_and_send_message() . . . . .	221
8.29.1.8 determine_unit() . . . . .	221
8.30 communication.h . . . . .	221
8.31 lib/comms/frame.cpp File Reference . . . . .	222
8.31.1 Detailed Description . . . . .	223
8.31.2 Typedef Documentation . . . . .	223
8.31.2.1 CommandHandler . . . . .	223
8.31.3 Variable Documentation . . . . .	223
8.31.3.1 eventRegister . . . . .	223
8.32 frame.cpp . . . . .	223
8.33 lib/comms/protocol.h File Reference . . . . .	225
8.33.1 Enumeration Type Documentation . . . . .	226
8.33.1.1 ErrorCode . . . . .	226
8.33.1.2 OperationType . . . . .	227
8.33.1.3 CommandAccessLevel . . . . .	227
8.33.1.4 ValueUnit . . . . .	227
8.33.1.5 ExceptionType . . . . .	228

---

8.33.1.6 Interface . . . . .	228
8.33.2 Function Documentation . . . . .	228
8.33.2.1 exception_type_to_string() . . . . .	228
8.33.2.2 error_code_to_string() . . . . .	229
8.33.2.3 operation_type_to_string() . . . . .	230
8.33.2.4 string_to_operation_type() . . . . .	231
8.33.2.5 hex_string_to_bytes() . . . . .	232
8.33.2.6 value_unit_type_to_string() . . . . .	232
8.33.3 Variable Documentation . . . . .	234
8.33.3.1 FRAME_BEGIN . . . . .	234
8.33.3.2 FRAME_END . . . . .	234
8.33.3.3 DELIMITER . . . . .	234
8.34 protocol.h . . . . .	234
8.35 lib/comms/receive.cpp File Reference . . . . .	235
8.35.1 Detailed Description . . . . .	236
8.35.2 Function Documentation . . . . .	236
8.35.2.1 on_receive() . . . . .	236
8.35.2.2 handle_uart_input() . . . . .	237
8.36 receive.cpp . . . . .	238
8.37 lib/comms/send.cpp File Reference . . . . .	239
8.37.1 Detailed Description . . . . .	239
8.37.2 Function Documentation . . . . .	239
8.37.2.1 send_message() . . . . .	239
8.37.2.2 send_frame_lora() . . . . .	240
8.37.2.3 send_frame_uart() . . . . .	240
8.37.2.4 split_and_send_message() . . . . .	240
8.38 send.cpp . . . . .	241
8.39 lib/comms/utils_converters.cpp File Reference . . . . .	241
8.39.1 Detailed Description . . . . .	242
8.39.2 Function Documentation . . . . .	242
8.39.2.1 exception_type_to_string() . . . . .	242
8.39.2.2 value_unit_type_to_string() . . . . .	243
8.39.2.3 operation_type_to_string() . . . . .	244
8.39.2.4 string_to_operation_type() . . . . .	245
8.39.2.5 error_code_to_string() . . . . .	246
8.39.2.6 hex_string_to_bytes() . . . . .	247
8.40 utils_converters.cpp . . . . .	248
8.41 lib/eventman/event_manager.cpp File Reference . . . . .	249
8.41.1 Detailed Description . . . . .	250
8.42 event_manager.cpp . . . . .	250
8.43 lib/eventman/event_manager.h File Reference . . . . .	251
8.43.1 Detailed Description . . . . .	253

---

8.43.2 Macro Definition Documentation . . . . .	253
8.43.2.1 EVENT_BUFFER_SIZE . . . . .	253
8.43.2.2 EVENT_FLUSH_THRESHOLD . . . . .	253
8.43.2.3 EVENT_LOG_FILE . . . . .	253
8.43.3 Function Documentation . . . . .	253
8.43.3.1 to_string() . . . . .	253
8.43.4 Variable Documentation . . . . .	253
8.43.4.1 id . . . . .	253
8.43.4.2 timestamp . . . . .	254
8.43.4.3 group . . . . .	254
8.43.4.4 event . . . . .	254
8.44 event_manager.h . . . . .	254
8.45 lib/location/gps_collector.cpp File Reference . . . . .	256
8.45.1 Macro Definition Documentation . . . . .	257
8.45.1.1 MAX_RAW_DATA_LENGTH . . . . .	257
8.45.2 Function Documentation . . . . .	257
8.45.2.1 splitString() . . . . .	257
8.45.2.2 collect_gps_data() . . . . .	258
8.45.3 Variable Documentation . . . . .	258
8.45.3.1 pause_gps_collection . . . . .	258
8.45.3.2 nmea_data . . . . .	258
8.46 gps_collector.cpp . . . . .	259
8.47 lib/location/gps_collector.h File Reference . . . . .	259
8.47.1 Function Documentation . . . . .	260
8.47.1.1 collect_gps_data() . . . . .	260
8.48 gps_collector.h . . . . .	261
8.49 lib/location/NMEA/NMEA_data.cpp File Reference . . . . .	261
8.49.1 Variable Documentation . . . . .	262
8.49.1.1 nmea_data . . . . .	262
8.50 NMEA_data.cpp . . . . .	262
8.51 lib/location/NMEA/NMEA_data.h File Reference . . . . .	263
8.51.1 Variable Documentation . . . . .	264
8.51.1.1 nmea_data . . . . .	264
8.52 NMEA_data.h . . . . .	264
8.53 lib/pin_config.cpp File Reference . . . . .	265
8.53.1 Variable Documentation . . . . .	265
8.53.1.1 lora_cs_pin . . . . .	265
8.53.1.2 lora_reset_pin . . . . .	265
8.53.1.3 lora_irq_pin . . . . .	265
8.53.1.4 lora_address_local . . . . .	266
8.53.1.5 lora_address_remote . . . . .	266
8.54 pin_config.cpp . . . . .	266

---

8.55 lib/pin_config.h File Reference . . . . .	266
8.55.1 Macro Definition Documentation . . . . .	267
8.55.1.1 DEBUG_UART_PORT . . . . .	267
8.55.1.2 DEBUG_UART_BAUD_RATE . . . . .	268
8.55.1.3 DEBUG_UART_TX_PIN . . . . .	268
8.55.1.4 DEBUG_UART_RX_PIN . . . . .	268
8.55.1.5 MAIN_I2C_PORT . . . . .	268
8.55.1.6 MAIN_I2C_SDA_PIN . . . . .	268
8.55.1.7 MAIN_I2C_SCL_PIN . . . . .	268
8.55.1.8 GPS_UART_PORT . . . . .	268
8.55.1.9 GPS_UART_BAUD_RATE . . . . .	268
8.55.1.10 GPS_UART_TX_PIN . . . . .	269
8.55.1.11 GPS_UART_RX_PIN . . . . .	269
8.55.1.12 GPS_POWER_ENABLE_PIN . . . . .	269
8.55.1.13 BUFFER_SIZE . . . . .	269
8.55.1.14 SD_SPI_PORT . . . . .	269
8.55.1.15 SD_MISO_PIN . . . . .	269
8.55.1.16 SD_MOSI_PIN . . . . .	269
8.55.1.17 SD_SCK_PIN . . . . .	269
8.55.1.18 SD_CS_PIN . . . . .	270
8.55.1.19 SD_CARD_DETECT_PIN . . . . .	270
8.55.1.20 SX1278_MISO . . . . .	270
8.55.1.21 SX1278_CS . . . . .	270
8.55.1.22 SX1278_SCK . . . . .	270
8.55.1.23 SX1278_MOSI . . . . .	270
8.55.1.24 SPI_PORT . . . . .	270
8.55.1.25 READ_BIT . . . . .	270
8.55.1.26 LORA_DEFAULT_SPI . . . . .	271
8.55.1.27 LORA_DEFAULT_SPI_FREQUENCY . . . . .	271
8.55.1.28 LORA_DEFAULT_SS_PIN . . . . .	271
8.55.1.29 LORA_DEFAULT_RESET_PIN . . . . .	271
8.55.1.30 LORA_DEFAULT_DIO0_PIN . . . . .	271
8.55.1.31 PA_OUTPUT_RFO_PIN . . . . .	271
8.55.1.32 PA_OUTPUT_PA_BOOST_PIN . . . . .	271
8.55.2 Variable Documentation . . . . .	272
8.55.2.1 lora_cs_pin . . . . .	272
8.55.2.2 lora_reset_pin . . . . .	272
8.55.2.3 lora_irq_pin . . . . .	272
8.55.2.4 lora_address_local . . . . .	272
8.55.2.5 lora_address_remote . . . . .	272
8.56 pin_config.h . . . . .	273
8.57 lib/powerman/INA3221/INA3221.cpp File Reference . . . . .	273

---

8.57.1 Detailed Description . . . . .	274
8.58 INA3221.cpp . . . . .	274
8.59 lib/powerman/INA3221/INA3221.h File Reference . . . . .	278
8.59.1 Detailed Description . . . . .	280
8.59.2 Enumeration Type Documentation . . . . .	280
8.59.2.1 ina3221_addr_t . . . . .	280
8.59.2.2 ina3221_ch_t . . . . .	280
8.59.2.3 ina3221_reg_t . . . . .	281
8.59.2.4 ina3221_conv_time_t . . . . .	281
8.59.2.5 ina3221_avg_mode_t . . . . .	282
8.59.3 Variable Documentation . . . . .	282
8.59.3.1 INA3221_CH_NUM . . . . .	282
8.59.3.2 SHUNT_VOLTAGE_LSB_UV . . . . .	282
8.60 INA3221.h . . . . .	283
8.61 lib/powerman/PowerManager.cpp File Reference . . . . .	285
8.62 PowerManager.cpp . . . . .	285
8.63 lib/powerman/PowerManager.h File Reference . . . . .	287
8.64 PowerManager.h . . . . .	288
8.65 lib/sensors/BH1750/BH1750.cpp File Reference . . . . .	289
8.66 BH1750.cpp . . . . .	289
8.67 lib/sensors/BH1750/BH1750.h File Reference . . . . .	290
8.67.1 Macro Definition Documentation . . . . .	291
8.67.1.1 _BH1750_DEVICE_ID . . . . .	291
8.67.1.2 _BH1750_MTREG_MIN . . . . .	291
8.67.1.3 _BH1750_MTREG_MAX . . . . .	292
8.67.1.4 _BH1750_DEFAULT_MTREG . . . . .	292
8.68 BH1750.h . . . . .	292
8.69 lib/sensors/BH1750/BH1750_WRAPPER.cpp File Reference . . . . .	292
8.70 BH1750_WRAPPER.cpp . . . . .	293
8.71 lib/sensors/BH1750/BH1750_WRAPPER.h File Reference . . . . .	294
8.72 BH1750_WRAPPER.h . . . . .	295
8.73 lib/sensors/BME280/BME280.cpp File Reference . . . . .	295
8.74 BME280.cpp . . . . .	296
8.75 lib/sensors/BME280/BME280.h File Reference . . . . .	299
8.76 BME280.h . . . . .	300
8.77 lib/sensors/BME280/BME280_WRAPPER.cpp File Reference . . . . .	302
8.78 BME280_WRAPPER.cpp . . . . .	302
8.79 lib/sensors/BME280/BME280_WRAPPER.h File Reference . . . . .	303
8.80 BME280_WRAPPER.h . . . . .	303
8.81 lib/sensors/HMC5883L/HMC5883L.cpp File Reference . . . . .	304
8.82 HMC5883L.cpp . . . . .	304
8.83 lib/sensors/HMC5883L/HMC5883L.h File Reference . . . . .	305

---

8.84 HMC5883L.h . . . . .	306
8.85 lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp File Reference . . . . .	306
8.86 HMC5883L_WRAPPER.cpp . . . . .	307
8.87 lib/sensors/HMC5883L/HMC5883L_WRAPPER.h File Reference . . . . .	308
8.88 HMC5883L_WRAPPER.h . . . . .	309
8.89 lib/sensors/ISensor.cpp File Reference . . . . .	309
8.89.1 Detailed Description . . . . .	309
8.90 ISensor.cpp . . . . .	310
8.91 lib/sensors/ISensor.h File Reference . . . . .	311
8.91.1 Enumeration Type Documentation . . . . .	312
8.91.1.1 SensorType . . . . .	312
8.91.1.2 SensorDataTypIdentifier . . . . .	312
8.92 ISensor.h . . . . .	313
8.93 lib/sensors/MPU6050/MPU6050.cpp File Reference . . . . .	313
8.94 MPU6050.cpp . . . . .	313
8.95 lib/sensors/MPU6050/MPU6050.h File Reference . . . . .	314
8.96 MPU6050.h . . . . .	314
8.97 lib/sensors/MPU6050/MPU6050_WRAPPER.cpp File Reference . . . . .	314
8.98 MPU6050_WRAPPER.cpp . . . . .	314
8.99 lib/sensors/MPU6050/MPU6050_WRAPPER.h File Reference . . . . .	314
8.100 MPU6050_WRAPPER.h . . . . .	315
8.101 lib/storage/storage.cpp File Reference . . . . .	316
8.101.1 Detailed Description . . . . .	316
8.101.2 Function Documentation . . . . .	317
8.101.2.1 fs_init() . . . . .	317
8.101.3 Variable Documentation . . . . .	318
8.101.3.1 sd_card_mounted . . . . .	318
8.102 storage.cpp . . . . .	318
8.103 lib/storage/storage.h File Reference . . . . .	318
8.103.1 Function Documentation . . . . .	319
8.103.1.1 fs_init() . . . . .	319
8.103.1.2 fs_open_file() . . . . .	320
8.103.1.3 fs_write_file() . . . . .	320
8.103.1.4 fs_read_file() . . . . .	321
8.103.1.5 fs_close_file() . . . . .	321
8.103.1.6 fs_file_exists() . . . . .	321
8.103.2 Variable Documentation . . . . .	321
8.103.2.1 sd_card_mounted . . . . .	321
8.104 storage.h . . . . .	321
8.105 lib/telemetry/telemetry_manager.cpp File Reference . . . . .	322
8.105.1 Detailed Description . . . . .	324
8.105.2 Macro Definition Documentation . . . . .	324

---

8.105.2.1 TELEMETRY_CSV_PATH . . . . .	324
8.105.2.2 DEFAULT_SAMPLE_INTERVAL_MS . . . . .	324
8.105.2.3 DEFAULT_FLUSH_THRESHOLD . . . . .	324
8.105.2.4 TELEMETRY_BUFFER_SIZE . . . . .	324
8.105.3 Variable Documentation . . . . .	324
8.105.3.1 powerManager . . . . .	324
8.105.3.2 systemClock . . . . .	325
8.105.3.3 nmea_data . . . . .	325
8.105.3.4 sample_interval_ms . . . . .	325
8.105.3.5 flush_threshold . . . . .	325
8.105.3.6 telemetry_buffer . . . . .	325
8.105.3.7 telemetry_buffer_count . . . . .	325
8.105.3.8 telemetry_buffer_write_index . . . . .	325
8.105.3.9 telemetry_mutex . . . . .	326
8.106 telemetry_manager.cpp . . . . .	326
8.107 lib/telemetry/telemetry_manager.h File Reference . . . . .	329
8.107.1 Detailed Description . . . . .	330
8.108 telemetry_manager.h . . . . .	331
8.109 lib/utils.cpp File Reference . . . . .	331
8.109.1 Detailed Description . . . . .	332
8.109.2 Function Documentation . . . . .	332
8.109.2.1 get_level_color() . . . . .	332
8.109.2.2 get_level_prefix() . . . . .	333
8.109.2.3 uart_print() . . . . .	335
8.109.2.4 base64_encode() . . . . .	336
8.109.2.5 calculate_checksum() . . . . .	337
8.109.3 Variable Documentation . . . . .	337
8.109.3.1 uart_mutex . . . . .	337
8.109.3.2 g_uart_verbosity . . . . .	337
8.109.3.3 base64_chars . . . . .	337
8.110 utils.cpp . . . . .	338
8.111 lib/utils.h File Reference . . . . .	339
8.111.1 Detailed Description . . . . .	340
8.111.2 Macro Definition Documentation . . . . .	341
8.111.2.1 ANSI_RED . . . . .	341
8.111.2.2 ANSI_GREEN . . . . .	341
8.111.2.3 ANSI_YELLOW . . . . .	341
8.111.2.4 ANSI_BLUE . . . . .	341
8.111.2.5 ANSI_CYAN . . . . .	341
8.111.2.6 ANSI_RESET . . . . .	341
8.111.3 Enumeration Type Documentation . . . . .	341
8.111.3.1 VerbosityLevel . . . . .	341

---

8.111.4 Function Documentation . . . . .	342
8.111.4.1 uart_print() . . . . .	342
8.111.4.2 base64_encode() . . . . .	344
8.111.4.3 calculate_checksum() . . . . .	345
8.111.5 Variable Documentation . . . . .	345
8.111.5.1 g_uart_verbosity . . . . .	345
8.112 utils.h . . . . .	345
8.113 main.cpp File Reference . . . . .	346
8.113.1 Macro Definition Documentation . . . . .	347
8.113.1.1 LOG_FILENAME . . . . .	347
8.113.2 Function Documentation . . . . .	347
8.113.2.1 process_pending_actions() . . . . .	347
8.113.2.2 core1_entry() . . . . .	347
8.113.2.3 init_systems() . . . . .	348
8.113.2.4 main() . . . . .	349
8.113.3 Variable Documentation . . . . .	350
8.113.3.1 powerManager . . . . .	350
8.113.3.2 systemClock . . . . .	350
8.113.3.3 pause_gps_collection . . . . .	350
8.113.3.4 buffer . . . . .	350
8.113.3.5 buffer_index . . . . .	350
8.114 main.cpp . . . . .	350
8.115 test/comms/test_comand_handlers.cpp File Reference . . . . .	352
8.115.1 Function Documentation . . . . .	353
8.115.1.1 send_frame_uart() . . . . .	353
8.115.1.2 send_frame_lora() . . . . .	354
8.115.1.3 setUp() . . . . .	354
8.115.1.4 tearDown() . . . . .	355
8.115.1.5 test_command_handler_get_operation() . . . . .	355
8.115.1.6 test_command_handler_set_operation() . . . . .	355
8.115.1.7 test_command_handler_invalid_operation() . . . . .	356
8.115.2 Variable Documentation . . . . .	356
8.115.2.1 uart_send_called . . . . .	356
8.115.2.2 lora_send_called . . . . .	357
8.115.2.3 last_frame_sent . . . . .	357
8.116 test_comand_handlers.cpp . . . . .	357
8.117 test/comms/test_converters.cpp File Reference . . . . .	358
8.117.1 Function Documentation . . . . .	358
8.117.1.1 test_operation_type_conversion() . . . . .	358
8.117.1.2 test_value_unit_type_conversion() . . . . .	359
8.117.1.3 test_exception_type_conversion() . . . . .	359
8.117.1.4 test_hex_string_conversion() . . . . .	360

---

8.118 test_converters.cpp . . . . .	360
8.119 test/comms/test_error_codes.cpp File Reference . . . . .	361
8.119.1 Function Documentation . . . . .	361
8.119.1.1 setUp() . . . . .	361
8.119.1.2 tearDown() . . . . .	362
8.119.1.3 test_error_code_conversion() . . . . .	362
8.120 test_error_codes.cpp . . . . .	362
8.121 test/comms/test_frame_build.cpp File Reference . . . . .	363
8.121.1 Function Documentation . . . . .	363
8.121.1.1 test_frame_build_success() . . . . .	363
8.121.1.2 test_frame_build_error() . . . . .	364
8.121.1.3 test_frame_build_info() . . . . .	364
8.122 test_frame_build.cpp . . . . .	365
8.123 test/comms/test_frame_coding.cpp File Reference . . . . .	365
8.123.1 Function Documentation . . . . .	366
8.123.1.1 test_frame_encode_basic() . . . . .	366
8.123.1.2 test_frame_decode_basic() . . . . .	366
8.123.1.3 test_frame_decode_invalid_header() . . . . .	367
8.124 test_frame_coding.cpp . . . . .	367
8.125 test/comms/test_frame_common.h File Reference . . . . .	368
8.125.1 Function Documentation . . . . .	369
8.125.1.1 create_test_frame() . . . . .	369
8.126 test_frame_common.h . . . . .	369
8.127 test/comms/test_frame_send.cpp File Reference . . . . .	370
8.127.1 Function Documentation . . . . .	370
8.127.1.1 setUp() . . . . .	370
8.127.1.2 tearDown() . . . . .	370
8.127.1.3 test_send_frame_uart() . . . . .	370
8.128 test_frame_send.cpp . . . . .	371
8.129 test/mocks/hardwareMocks.cpp File Reference . . . . .	371
8.129.1 Function Documentation . . . . .	372
8.129.1.1 mock_uart_puts() . . . . .	372
8.129.1.2 mock_uart_init() . . . . .	372
8.129.1.3 mock_spi_write_blocking() . . . . .	372
8.129.1.4 mock_spi_read_blocking() . . . . .	372
8.129.2 Variable Documentation . . . . .	373
8.129.2.1 mock_uart_enabled . . . . .	373
8.129.2.2 uart_output_buffer . . . . .	373
8.129.2.3 mock_spi_enabled . . . . .	373
8.129.2.4 spi_output_buffer . . . . .	373
8.130 hardwareMocks.cpp . . . . .	373
8.131 test/mocks/hardwareMocks.h File Reference . . . . .	374

---

8.131.1 Function Documentation . . . . .	375
8.131.1.1 mock_uart_puts() . . . . .	375
8.131.1.2 mock_uart_init() . . . . .	375
8.131.1.3 mock_spi_write_blocking() . . . . .	375
8.131.1.4 mock_spi_read_blocking() . . . . .	375
8.131.2 Variable Documentation . . . . .	375
8.131.2.1 mock_uart_enabled . . . . .	375
8.131.2.2 uart_output_buffer . . . . .	375
8.131.2.3 mock_spi_enabled . . . . .	376
8.131.2.4 spi_output_buffer . . . . .	376
8.132 hardwareMocks.h . . . . .	376
8.133 test/test_runner.cpp File Reference . . . . .	376
8.133.1 Function Documentation . . . . .	377
8.133.1.1 test_frame_encode_basic() . . . . .	377
8.133.1.2 test_frame_decode_basic() . . . . .	378
8.133.1.3 test_frame_decode_invalid_header() . . . . .	378
8.133.1.4 test_frame_build_success() . . . . .	379
8.133.1.5 test_frame_build_error() . . . . .	379
8.133.1.6 test_frame_build_info() . . . . .	380
8.133.1.7 test_operation_type_conversion() . . . . .	381
8.133.1.8 test_value_unit_type_conversion() . . . . .	381
8.133.1.9 test_exception_type_conversion() . . . . .	382
8.133.1.10 test_hex_string_conversion() . . . . .	383
8.133.1.11 test_command_handler_get_operation() . . . . .	383
8.133.1.12 test_command_handler_set_operation() . . . . .	384
8.133.1.13 test_command_handler_invalid_operation() . . . . .	385
8.133.1.14 test_error_code_conversion() . . . . .	385
8.133.1.15 main() . . . . .	386
8.134 test_runner.cpp . . . . .	386
Index . . . . .	389

# Chapter 1

## Clock Commands

Member `handle_clock_sync_interval (const std::string &param, OperationType operationType)`

Command ID: 3.3

Member `handle_enable_gps_uart_passthrough (const std::string &param, OperationType operationType)`

Command ID: 7.2

Member `handle_enter_bootloader_mode (const std::string &param, OperationType operationType)`

Command ID: 2

Member `handle_get_build_version (const std::string &param, OperationType operationType)`

Command ID: 1

Member `handle_get_commands_list (const std::string &param, OperationType operationType)`

Command ID: 0

Member `handle_get_current_charge_solar (const std::string &param, OperationType operationType)`

Command ID: 2.5

Member `handle_get_current_charge_total (const std::string &param, OperationType operationType)`

Command ID: 2.6

Member `handle_get_current_charge_usb (const std::string &param, OperationType operationType)`

Command ID: 2.4

Member `handle_get_current_draw (const std::string &param, OperationType operationType)`

Command ID: 2.7

Member `handle_get_event_count (const std::string &param, OperationType operationType)`

Command ID: 5.2

Member `handle_get_gga_data (const std::string &param, OperationType operationType)`

Command ID: 7.4

Member `handle_get_last_events (const std::string &param, OperationType operationType)`

Command ID: 5.1

Member `handle_get_last_sync_time (const std::string &param, OperationType operationType)`

Command ID: 3.7

Member `handle_get_power_manager_ids (const std::string &param, OperationType operationType)`

Command ID: 2.0

Member `handle_get_rmc_data (const std::string &param, OperationType operationType)`

Command ID: 7.3

Member `handle_get_sensor_data (const std::string &param, OperationType operationType)`

Command ID: 3.0

Member `handle_get_sensor_list (const std::string &param, OperationType operationType)`

Command ID: 4.2

Member `handle_get_voltage_5v (const std::string &param, OperationType operationType)`

Command ID: 2.3

Member `handle_get_voltage_battery (const std::string &param, OperationType operationType)`

Command ID: 2.2

Member `handle_gps_power_status (const std::string &param, OperationType operationType)`

Command ID: 7.1

Member `handle_list_files (const std::string &param, OperationType operationType)`

Command ID: 6.0

Member `handle_mount (const std::string &param, OperationType operationType)`

Command ID: 6.4

Member `handle_sensor_config (const std::string &param, OperationType operationType)`

Command ID: 3.1

Member `handle_time (const std::string &param, OperationType operationType)`

Command ID: 3.0

Member `handle_timezone_offset (const std::string &param, OperationType operationType)`

Command ID: 3.1

Member `handle_verbosity (const std::string &param, OperationType operationType)`

Command ID: 1.8

# Chapter 2

## Topic Index

### 2.1 Topics

Here is a list of all topics with brief descriptions:

Clock Management Commands . . . . .	11
Command System . . . . .	15
Diagnostic Commands . . . . .	17
Event Commands . . . . .	21
GPS Commands . . . . .	24
Power Commands . . . . .	28
Sensor Commands . . . . .	34
Storage Commands . . . . .	37
Frame Handling . . . . .	39
Event Manager . . . . .	44
INA3221 Power Monitor . . . . .	51
Configuration Functions . . . . .	51
Measurement Functions . . . . .	61
Alert Functions . . . . .	63
Telemetry Manager . . . . .	69



# Chapter 3

## Hierarchical Index

### 3.1 Class Hierarchy

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

BH1750 . . . . .	75
BME280 . . . . .	81
BME280CalibParam . . . . .	90
INA3221::conf_reg_t . . . . .	96
DS3231 . . . . .	98
ds3231_data_t . . . . .	117
EventEmitter . . . . .	119
EventLog . . . . .	121
EventManager . . . . .	123
EventManagerImpl . . . . .	128
FileHandle . . . . .	131
Frame . . . . .	132
HMC5883L . . . . .	134
INA3221 . . . . .	140
ISensor . . . . .	146
BH1750Wrapper . . . . .	78
BME280Wrapper . . . . .	94
HMC5883LWrapper . . . . .	137
MPU6050Wrapper . . . . .	151
INA3221::masken_reg_t . . . . .	148
NMEAData . . . . .	153
PowerManager . . . . .	156
SensorWrapper . . . . .	162
TelemetryRecord . . . . .	167



# Chapter 4

## Class Index

### 4.1 Class List

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

BH1750 . . . . .	75
BH1750Wrapper . . . . .	78
BME280 . . . . .	81
BME280CalibParam . . . . .	90
BME280Wrapper . . . . .	94
INA3221::conf_reg_t Configuration register bit fields . . . . .	96
DS3231 . . . . .	98
Class for interfacing with the DS3231 real-time clock . . . . .	98
ds3231_data_t . . . . .	117
Structure to hold time and date information from DS3231 . . . . .	117
EventEmitter . . . . .	119
Provides a static method for emitting events . . . . .	119
EventLog . . . . .	121
Represents a single event log entry . . . . .	121
EventManager . . . . .	123
Manages the event logging system . . . . .	123
EventManagerImpl . . . . .	128
Implementation of the EventManager class . . . . .	128
FileHandle . . . . .	131
Frame . . . . .	132
Represents a communication frame used for data exchange . . . . .	132
HMC5883L . . . . .	134
HMC5883LWrapper . . . . .	137
INA3221 . . . . .	140
INA3221 Triple-Channel Power Monitor driver class . . . . .	140
ISensor . . . . .	146
INA3221::masken_reg_t Mask/Enable register bit fields . . . . .	148
MPU6050Wrapper . . . . .	151
NMEAData . . . . .	153
PowerManager . . . . .	156
SensorWrapper . . . . .	162
Manages different sensor types and provides a unified interface for accessing sensor data . . . . .	162
TelemetryRecord . . . . .	167



# Chapter 5

## File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

<code>build_number.h</code> . . . . .	173
<code>includes.h</code> . . . . .	174
<code>main.cpp</code> . . . . .	346
<code>lib/pin_config.cpp</code> . . . . .	265
<code>lib/pin_config.h</code> . . . . .	266
<code>lib/utils.cpp</code> Implementation of utility functions for the Kabisat firmware . . . . .	331
<code>lib/utils.h</code> . . . . .	339
Utility functions and definitions for the Kabisat firmware . . . . .	339
<code>lib/clock/DS3231.cpp</code> . . . . .	175
<code>lib/clock/DS3231.h</code> . . . . .	180
<code>lib/comms/communication.cpp</code> . . . . .	213
<code>lib/comms/communication.h</code> . . . . .	216
<code>lib/comms/frame.cpp</code> Implements functions for encoding, decoding, building, and processing Frames . . . . .	222
<code>lib/comms/protocol.h</code> . . . . .	225
<code>lib/comms/receive.cpp</code> Implements functions for receiving and processing data, including LoRa and UART input . . . . .	235
<code>lib/comms/send.cpp</code> Implements functions for sending data, including LoRa messages and Frames . . . . .	239
<code>lib/comms/utils_converters.cpp</code> Implements utility functions for converting between different data types . . . . .	241
<code>lib/comms/commands/clock_commands.cpp</code> . . . . .	185
<code>lib/comms/commands/commands.cpp</code> . . . . .	189
<code>lib/comms/commands/commands.h</code> . . . . .	191
<code>lib/comms/commands/diagnostic_commands.cpp</code> . . . . .	193
<code>lib/comms/commands/event_commands.cpp</code> . . . . .	196
<code>lib/comms/commands/gps_commands.cpp</code> . . . . .	197
<code>lib/comms/commands/power_commands.cpp</code> . . . . .	201
<code>lib/comms/commands/sensor_commands.cpp</code> . . . . .	205
<code>lib/comms/commands/storage_commands.cpp</code> . . . . .	210
<code>lib/eventman/event_manager.cpp</code> Implements the event management system for the Kabisat firmware . . . . .	249
<code>lib/eventman/event_manager.h</code> . . . . .	251
Manages the event logging system for the Kabisat firmware . . . . .	251

lib/location/gps_collector.cpp . . . . .	256
lib/location/gps_collector.h . . . . .	259
lib/location/NMEA/NMEA_data.cpp . . . . .	261
lib/location/NMEA/NMEA_data.h . . . . .	263
lib/powerman/PowerManager.cpp . . . . .	285
lib/powerman/PowerManager.h . . . . .	287
lib/powerman/INA3221/INA3221.cpp Implementation of the <a href="#">INA3221</a> power monitor driver . . . . .	273
lib/powerman/INA3221/INA3221.h Header file for the <a href="#">INA3221</a> triple-channel power monitor driver . . . . .	278
lib/sensors/ISensor.cpp Implements the <a href="#">SensorWrapper</a> class for managing different sensor types . . . . .	309
lib/sensors/ISensor.h . . . . .	311
lib/sensors/BH1750/BH1750.cpp . . . . .	289
lib/sensors/BH1750/BH1750.h . . . . .	290
lib/sensors/BH1750/BH1750_WRAPPER.cpp . . . . .	292
lib/sensors/BH1750/BH1750_WRAPPER.h . . . . .	294
lib/sensors/BME280/BME280.cpp . . . . .	295
lib/sensors/BME280/BME280.h . . . . .	299
lib/sensors/BME280/BME280_WRAPPER.cpp . . . . .	302
lib/sensors/BME280/BME280_WRAPPER.h . . . . .	303
lib/sensors/HMC5883L/HMC5883L.cpp . . . . .	304
lib/sensors/HMC5883L/HMC5883L.h . . . . .	305
lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp . . . . .	306
lib/sensors/HMC5883L/HMC5883L_WRAPPER.h . . . . .	308
lib/sensors/MPU6050/MPU6050.cpp . . . . .	313
lib/sensors/MPU6050/MPU6050.h . . . . .	314
lib/sensors/MPU6050/MPU6050_WRAPPER.cpp . . . . .	314
lib/sensors/MPU6050/MPU6050_WRAPPER.h . . . . .	314
lib/storage/storage.cpp Implements file system operations for the Kubisat firmware . . . . .	316
lib/storage/storage.h . . . . .	318
lib/telemetry/telemetry_manager.cpp Implementation of telemetry collection and storage functionality . . . . .	322
lib/telemetry/telemetry_manager.h System telemetry collection and logging . . . . .	329
test/test_runner.cpp . . . . .	376
test/comms/test_command_handlers.cpp . . . . .	352
test/comms/test_converters.cpp . . . . .	358
test/comms/test_error_codes.cpp . . . . .	361
test/comms/test_frame_build.cpp . . . . .	363
test/comms/test_frame_coding.cpp . . . . .	365
test/comms/test_frame_common.h . . . . .	368
test/comms/test_frame_send.cpp . . . . .	370
test/mock/hardware_mock.cpp . . . . .	371
test/mock/hardware_mock.h . . . . .	374

# Chapter 6

## Topic Documentation

### 6.1 Clock Management Commands

Commands for managing system time and clock settings.

#### Functions

- std::vector< Frame > handle\_time (const std::string &param, OperationType operationType)  
*Handler for getting and setting system time.*
- std::vector< Frame > handle\_timezone\_offset (const std::string &param, OperationType operationType)  
*Handler for getting and setting timezone offset.*
- std::vector< Frame > handle\_clock\_sync\_interval (const std::string &param, OperationType operationType)  
*Handler for getting and setting clock synchronization interval.*
- std::vector< Frame > handle\_get\_last\_sync\_time (const std::string &param, OperationType operationType)  
*Handler for getting last clock sync time.*

#### Variables

- DS3231 systemClock

#### 6.1.1 Detailed Description

Commands for managing system time and clock settings.

#### 6.1.2 Function Documentation

##### 6.1.2.1 handle\_time()

```
std::vector< Frame > handle_time (
    const std::string & param,
    OperationType operationType)
```

Handler for getting and setting system time.

### Parameters

<i>param</i>	For SET: Unix timestamp as string, for GET: empty string
<i>operationType</i>	GET/SET

### Returns

Vector of frames containing success/error and current time or confirmation

### Note

GET: **KBST;0;GET;3;0;;KBST**

When getting time, returns format "HH:MM:SS Weekday DD.MM.YYYY"

SET: **KBST;0;SET;3;0;TIMESTAMP;KBST**

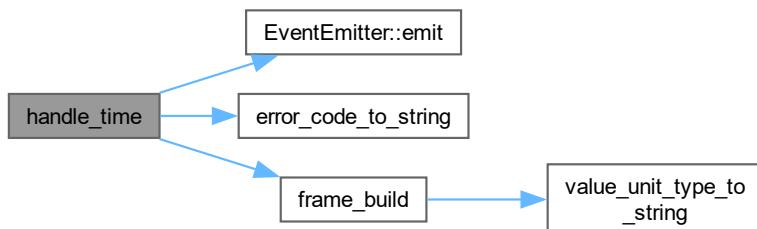
When setting time, expects Unix timestamp as parameter

### Command

Command ID: 3.0

Definition at line 32 of file [clock\\_commands.cpp](#).

Here is the call graph for this function:



### 6.1.2.2 handle\_timezone\_offset()

```
std::vector< Frame > handle_timezone_offset (
    const std::string & param,
    OperationType operationType)
```

Handler for getting and setting timezone offset.

### Parameters

<i>param</i>	For SET: Timezone offset in minutes (-720 to +720), for GET: empty string
<i>operationType</i>	GET/SET

**Returns**

Vector of frames containing success/error and timezone offset in minutes

**Note**

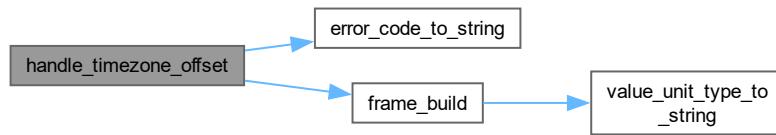
GET: **KBST;0;GET;3;1;;KBST**

SET: **KBST;0;SET;3;1;OFFSET;KBST**

**Command** Command ID: 3.1

Definition at line 97 of file [clock\\_commands.cpp](#).

Here is the call graph for this function:

**6.1.2.3 handle\_clock\_sync\_interval()**

```
std::vector< Frame > handle_clock_sync_interval (
    const std::string & param,
    OperationType operationType)
```

Handler for getting and setting clock synchronization interval.

**Parameters**

<code>param</code>	For SET: Sync interval in seconds, for GET: empty string
<code>operationType</code>	GET/SET

**Returns**

Vector with frame containing success/error and sync interval in seconds

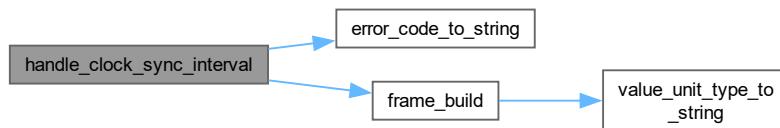
### Note

**GET:** **KBST;0;GET;3;3;;KBST**  
**SET:** **KBST;0;SET;3;3;INTERVAL;KBST**

**Command** Command ID: 3.3

Definition at line 158 of file [clock\\_commands.cpp](#).

Here is the call graph for this function:



### 6.1.2.4 handle\_get\_last\_sync\_time()

```
std::vector< Frame > handle_get_last_sync_time (
    const std::string & param,
    OperationType operationType)
```

Handler for getting last clock sync time.

#### Parameters

<code>param</code>	Empty string expected
<code>operationType</code>	GET

#### Returns

Vector with one frame containing success/error and last sync time as Unix timestamp

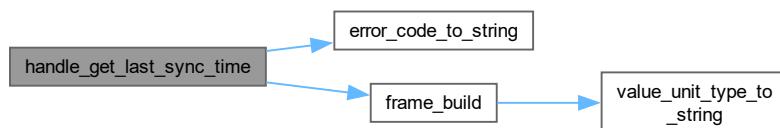
### Note

**KBST;0;GET;3;7;;KBST**

**Command** Command ID: 3.7

Definition at line 216 of file [clock\\_commands.cpp](#).

Here is the call graph for this function:



### 6.1.3 Variable Documentation

#### 6.1.3.1 systemClock

```
DS3231 systemClock [extern]
```

## 6.2 Command System

Core command system implementation.

### Typedefs

- using `CommandHandler` = `std::function<std::vector<Frame>(const std::string&, OperationType)>`  
*Function type for command handlers.*
- using `CommandMap` = `std::map<uint32_t, CommandHandler>`  
*Map type for storing command handlers.*

### Functions

- `std::vector< Frame > execute_command (uint32_t commandKey, const std::string &param, OperationType operationType)`  
*Executes a command based on its key.*

### Variables

- `CommandMap command_handlers`  
*Global map of all command handlers.*

### 6.2.1 Detailed Description

Core command system implementation.

## 6.2.2 Typedef Documentation

### 6.2.2.1 CommandHandler

```
using CommandHandler = std::function<std::vector<Frame>(const std::string&, OperationType)>
```

Function type for command handlers.

Definition at line 15 of file `commands.cpp`.

### 6.2.2.2 CommandMap

```
using CommandMap = std::map<uint32_t, CommandHandler>
```

Map type for storing command handlers.

Definition at line 21 of file [commands.cpp](#).

## 6.2.3 Function Documentation

### 6.2.3.1 execute\_command()

```
std::vector< Frame > execute_command (
    uint32_t commandKey,
    const std::string & param,
    OperationType operationType)
```

Executes a command based on its key.

#### Parameters

<i>commandKey</i>	Combined group and command ID (group << 8   command)
<i>param</i>	Command parameter string
<i>operationType</i>	Operation type (GET/SET)

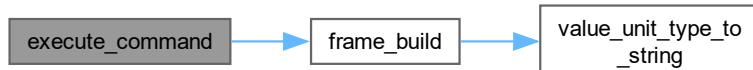
#### Returns

[Frame](#) Response frame containing execution result

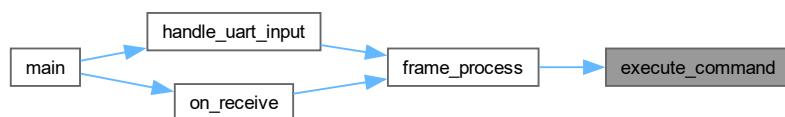
Looks up the command handler in commandHandlers map and executes it

Definition at line 65 of file [commands.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.2.4 Variable Documentation

### 6.2.4.1 command\_handlers

```
CommandMap command_handlers
```

#### Initial value:

```
= {
    {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(0)), handle_get_commands_list},
    {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(1)), handle_get_build_version},
    {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(8)), handle_verbosity},
    {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(9)), handle_enter_bootloader_mode},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(0)), handle_get_power_manager_ids},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(2)), handle_get_voltage_battery},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(3)), handle_get_voltage_5v},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(4)), handle_get_current_charge_usb},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(5)), handle_get_current_charge_solar},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(6)), handle_get_current_charge_total},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(7)), handle_get_current_draw},
    {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(0)), handle_time},
    {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(1)), handle_timezone_offset},
    {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(2)), handle_clock_sync_interval},
    {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(3)), handle_get_last_sync_time},
    {((static_cast<uint32_t>(4) << 8) | static_cast<uint32_t>(0)), handle_get_sensor_data},
    {((static_cast<uint32_t>(4) << 8) | static_cast<uint32_t>(1)), handle_sensor_config},
    {((static_cast<uint32_t>(4) << 8) | static_cast<uint32_t>(3)), handle_get_sensor_list},
    {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(1)), handle_get_last_events},
    {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(2)), handle_get_event_count},
    {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(0)), handle_list_files},
    {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(4)), handle_mount},
    {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(1)), handle_gps_power_status},
    {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(2)), handle_enable_gps_uart_passthrough},
    {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(3)), handle_get_rmc_data},
    {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(4)), handle_get_gga_data},
}
```

Global map of all command handlers.

Maps command keys (group << 8 | command) to their handler functions

Definition at line 27 of file [commands.cpp](#).

## 6.3 Diagnostic Commands

### Functions

- std::vector< [Frame](#) > [handle\\_get\\_commands\\_list](#) (const std::string &param, [OperationType](#) operationType)  
*Handler for listing all available commands on UART.*
- std::vector< [Frame](#) > [handle\\_get\\_build\\_version](#) (const std::string &param, [OperationType](#) operationType)  
*Get firmware build version.*
- std::vector< [Frame](#) > [handle\\_verbosity](#) (const std::string &param, [OperationType](#) operationType)  
*Handles setting or getting the UART verbosity level.*
- std::vector< [Frame](#) > [handle\\_enter\\_bootloader\\_mode](#) (const std::string &param, [OperationType](#) operationType)  
*Reboot system to USB firmware loader.*

### Variables

- volatile bool [g\\_pending\\_bootloader\\_reset](#)

### 6.3.1 Detailed Description

### 6.3.2 Function Documentation

#### 6.3.2.1 handle\_get\_commands\_list()

```
std::vector< Frame > handle_get_commands_list (
    const std::string & param,
    OperationType operationType)
```

Handler for listing all available commands on UART.

#### Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	GET

#### Returns

Vector of response frames - start frame, sequence of elements, end frame

#### Note

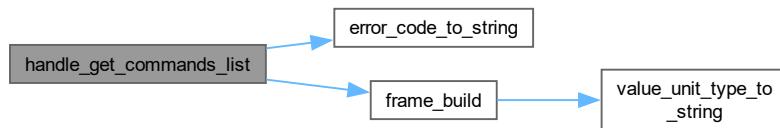
**KBST;0;GET;1;0;;TSBK**

Print all available commands on UART port

**Command** Command ID: 0

Definition at line 23 of file [diagnostic\\_commands.cpp](#).

Here is the call graph for this function:



#### 6.3.2.2 handle\_get\_build\_version()

```
std::vector< Frame > handle_get_build_version (
    const std::string & param,
    OperationType operationType)
```

Get firmware build version.

## Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	GET

## Returns

One-element vector with result frame

## Note

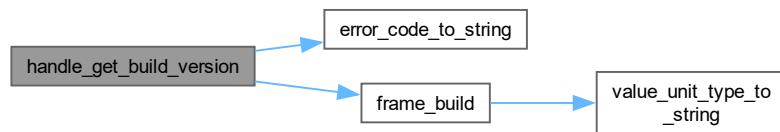
**KBST;0;GET;1;1;;TSBK**

Get the firmware build version

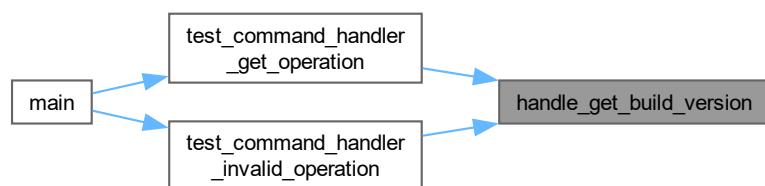
**Command** Command ID: 1

Definition at line 77 of file [diagnostic\\_commands.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.3 handle\_verbosity()

```
std::vector< Frame > handle_verbosity (
    const std::string & param,
    OperationType operationType)
```

Handles setting or getting the UART verbosity level.

This function allows the user to either retrieve the current UART verbosity level or set a new verbosity level.

### Parameters

<i>param</i>	The desired verbosity level (0-5) as a string. If empty, the current level is returned.
<i>operationType</i>	The operation type. Must be GET to retrieve the current level, or SET to set a new level.

### Returns

Vector containing one frame indicating the result of the operation.

- Success (GET): Frame containing the current verbosity level.
- Success (SET): Frame with "LEVEL SET" message.
- Error: Frame with error message (e.g., "INVALID LEVEL (0-5)", "INVALID FORMAT").

### Note

**KBST;0;GET;1;8;;TSBK** - Gets the current verbosity level.

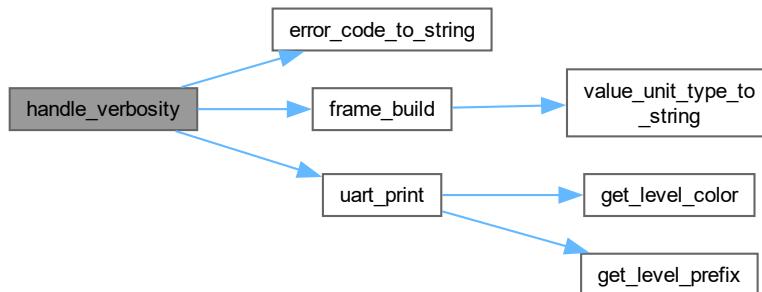
**KBST;0;SET;1;8;[level];TSBK** - Sets the verbosity level.

Example: **KBST;0;SET;1;8;2;TSBK** - Sets the verbosity level to 2.

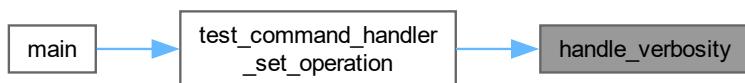
**Command** Command ID: 1.8

Definition at line 119 of file [diagnostic\\_commands.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.3.2.4 handle\_enter\_bootloader\_mode()

```
std::vector< Frame > handle_enter_bootloader_mode (
    const std::string & param,
    OperationType operationType)
```

Reboot system to USB firmware loader.

#### Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	Must be SET

#### Returns

Frame with operation result

#### Note

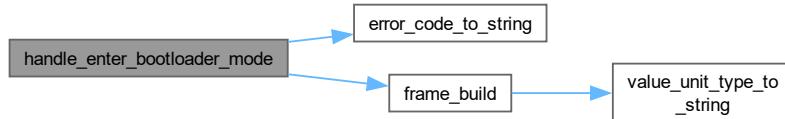
**KBST;0;SET;1;9;;TSBK**

Reboot the system to USB firmware loader

**Command** Command ID: 2

Definition at line 158 of file [diagnostic\\_commands.cpp](#).

Here is the call graph for this function:



## 6.3.3 Variable Documentation

### 6.3.3.1 g\_pending\_bootloader\_reset

```
volatile bool g_pending_bootloader_reset [extern]
```

Definition at line 7 of file [main.cpp](#).

## 6.4 Event Commands

Commands for accessing and managing system event logs.

## Functions

- std::vector< [Frame](#) > [handle\\_get\\_last\\_events](#) (const std::string &param, [OperationType](#) operationType)  
*Handler for retrieving last N events from the event log.*
- std::vector< [Frame](#) > [handle\\_get\\_event\\_count](#) (const std::string &param, [OperationType](#) operationType)  
*Handler for getting total number of events in the log.*

### 6.4.1 Detailed Description

Commands for accessing and managing system event logs.

### 6.4.2 Function Documentation

#### 6.4.2.1 [handle\\_get\\_last\\_events\(\)](#)

```
std::vector< Frame > handle_get_last_events (
    const std::string & param,
    OperationType operationType)
```

Handler for retrieving last N events from the event log.

#### Parameters

<i>param</i>	Number of events to retrieve (optional, default 10). If 0, all events are returned.
<i>operationType</i>	GET

#### Returns

[Frame](#) containing:

- Success: A sequence of frames, each containing up to 10 hex-encoded events. Each event is in the format IIIITTTTTTGGE, separated by '-'.
  - III: Event ID (16-bit, 4 hex characters)
  - TTTTTTTT: Unix Timestamp (32-bit, 8 hex characters)
  - GG: Event Group (8-bit, 2 hex characters)
  - EE: Event Type (8-bit, 2 hex characters) The last frame in the sequence is a VAL frame with the message "SEQ\_DONE".
- Error: A single frame with an error message:
  - "INVALID OPERATION": If the operation type is not GET.
  - "INVALID COUNT": If the count is greater than EVENT\_BUFFER\_SIZE.
  - "INVALID PARAMETER": If the parameter is not a valid unsigned integer.

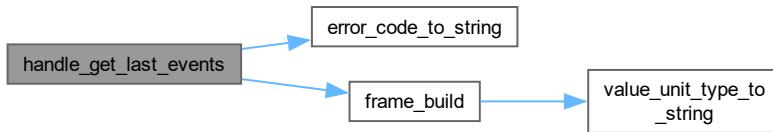
**Note**

**KBST;0;GET;5;1;[N];TSBK** - Retrieves the last N events. If N is 0, retrieves all events.  
Returns up to 10 most recent events per frame.

**Command** Command ID: 5.1

Definition at line 33 of file [event\\_commands.cpp](#).

Here is the call graph for this function:

**6.4.2.2 handle\_get\_event\_count()**

```
std::vector< Frame > handle_get_event_count (
    const std::string & param,
    OperationType operationType)
```

Handler for getting total number of events in the log.

**Parameters**

<i>param</i>	Empty string expected
<i>operationType</i>	GET

**Returns**

**Frame** containing:

- Success: Number of events currently in the log
- Error: "INVALID REQUEST"

**Note**

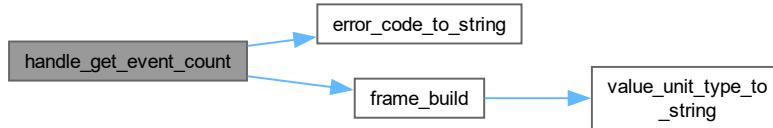
**KBST;0;GET;5;2;;TSBK**

Returns the total number of events in the log

**Command** Command ID: 5.2

Definition at line 100 of file [event\\_commands.cpp](#).

Here is the call graph for this function:



## 6.5 GPS Commands

Commands for controlling and monitoring the GPS module.

### Functions

- std::vector< [Frame](#) > [handle\\_gps\\_power\\_status](#) (const std::string &param, [OperationType](#) operationType)  
*Handler for controlling GPS module power state.*
- std::vector< [Frame](#) > [handle\\_enable\\_gps\\_uart\\_passthrough](#) (const std::string &param, [OperationType](#) operationType)  
*Handler for enabling GPS transparent mode (UART pass-through)*
- std::vector< [Frame](#) > [handle\\_get\\_rmc\\_data](#) (const std::string &param, [OperationType](#) operationType)  
*Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.*
- std::vector< [Frame](#) > [handle\\_get\\_gga\\_data](#) (const std::string &param, [OperationType](#) operationType)  
*Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.*

### 6.5.1 Detailed Description

Commands for controlling and monitoring the GPS module.

### 6.5.2 Function Documentation

#### 6.5.2.1 [handle\\_gps\\_power\\_status\(\)](#)

```
std::vector< Frame > handle_gps_power_status (
    const std::string & param,
    OperationType operationType)
```

Handler for controlling GPS module power state.

#### Parameters

<i>param</i>	For SET: "0" to power off, "1" to power on. For GET: empty
<i>operationType</i>	GET to read current state, SET to change state

**Returns**

Vector of Frames containing:

- Success: Current power state (0/1) or
- Error: Error reason

**Note**

**KBST;0;GET;7;1;;TSBK**

Return current GPS module power state: ON/OFF

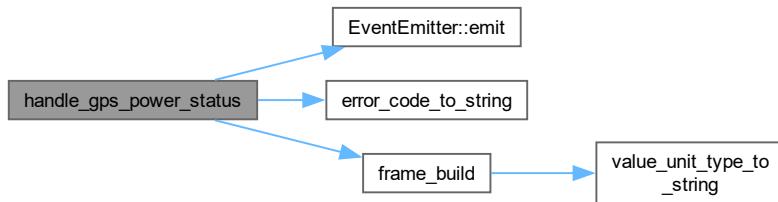
**KBST;0;SET;7;1;POWER;TSBK**

POWER - 0 - OFF, 1 - ON

**Command** Command ID: 7.1

Definition at line 32 of file [gps\\_commands.cpp](#).

Here is the call graph for this function:

**6.5.2.2 handle\_enable\_gps\_uart\_passthrough()**

```
std::vector< Frame > handle_enable_gps_uart_passthrough (
    const std::string & param,
    OperationType operationType)
```

Handler for enabling GPS transparent mode (UART pass-through)

**Parameters**

<code>param</code>	TIMEOUT in seconds (optional, defaults to 60)
<code>operationType</code>	SET

**Returns**

Vector of Frames containing:

- Success: Exit message + reason or
- Error: Error reason

**Note****KBST;0;SET;7;2;TIMEOUT;TSBK**

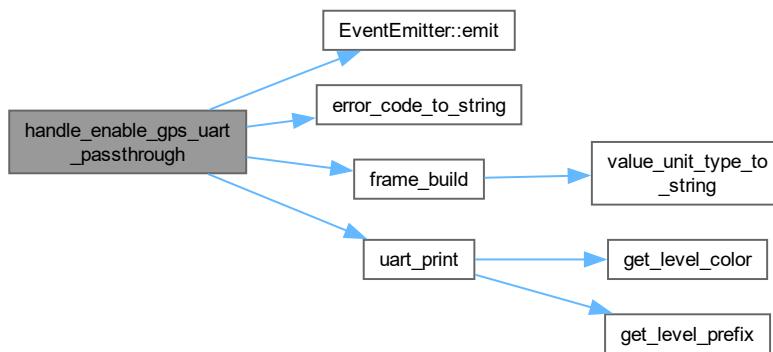
TIMEOUT - 1-600s, default 60s

Enters a pass-through mode where UART communication is bridged directly to GPS

Send "##EXIT##" to exit mode before TIMEOUT

**Command** Command ID: 7.2Definition at line 89 of file [gps\\_commands.cpp](#).

Here is the call graph for this function:

**6.5.2.3 handle\_get\_rmc\_data()**

```
std::vector< Frame > handle_get_rmc_data (
    const std::string & param,
    OperationType operationType)
```

Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.

**Parameters**

<code>param</code>	Empty string expected
<code>operationType</code>	GET

**Returns**

Vector of Frames containing:

- Success: Comma-separated RMC tokens or
- Error: Error message

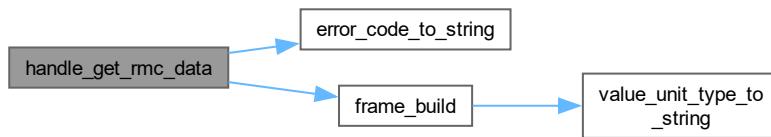
Note

**KBST;0;GET;7;3;;TSBK**

**Command** Command ID: 7.3

Definition at line 193 of file [gps\\_commands.cpp](#).

Here is the call graph for this function:



#### 6.5.2.4 handle\_get\_gga\_data()

```
std::vector< Frame > handle_get_gga_data (
    const std::string & param,
    OperationType operationType)
```

Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.

Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	GET

Returns

Vector of Frames containing:

- Success: Comma-separated GGA tokens or
- Error: Error message

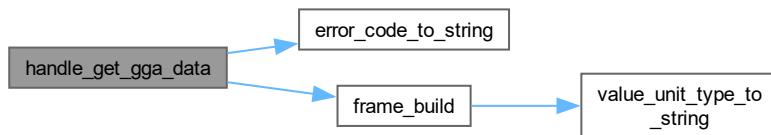
Note

**KBST;0;GET;7;4;;TSBK**

**Command** Command ID: 7.4

Definition at line 235 of file [gps\\_commands.cpp](#).

Here is the call graph for this function:



## 6.6 Power Commands

Commands for monitoring power subsystem and battery management.

### Functions

- std::vector< Frame > handle\_get\_power\_manager\_ids (const std::string &param, OperationType operationType)
   
*Handler for retrieving Power Manager IDs.*
- std::vector< Frame > handle\_get\_voltage\_battery (const std::string &param, OperationType operationType)
   
*Handler for getting battery voltage.*
- std::vector< Frame > handle\_get\_voltage\_5v (const std::string &param, OperationType operationType)
   
*Handler for getting 5V rail voltage.*
- std::vector< Frame > handle\_get\_current\_charge\_usb (const std::string &param, OperationType operationType)
   
*Handler for getting USB charge current.*
- std::vector< Frame > handle\_get\_current\_charge\_solar (const std::string &param, OperationType operationType)
   
*Handler for getting solar panel charge current.*
- std::vector< Frame > handle\_get\_current\_charge\_total (const std::string &param, OperationType operationType)
   
*Handler for getting total charge current.*
- std::vector< Frame > handle\_get\_current\_draw (const std::string &param, OperationType operationType)
   
*Handler for getting system current draw.*

### 6.6.1 Detailed Description

Commands for monitoring power subsystem and battery management.

### 6.6.2 Function Documentation

#### 6.6.2.1 handle\_get\_power\_manager\_ids()

```
std::vector< Frame > handle_get_power_manager_ids (
    const std::string & param,
    OperationType operationType)
```

Handler for retrieving Power Manager IDs.

#### Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	GET

**Returns**

Vector of Frames containing:

- Success: String of Power Manager IDs
- Error: Error message

**Note**

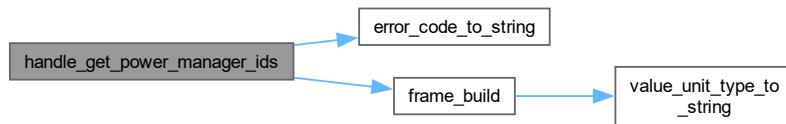
**KBST;0;GET;2;0;;TSBK**

This command is used to retrieve the IDs of the Power Manager

**Command** Command ID: 2.0

Definition at line 30 of file [power\\_commands.cpp](#).

Here is the call graph for this function:

**6.6.2.2 handle\_get\_voltage\_battery()**

```
std::vector< Frame > handle_get_voltage_battery (
    const std::string & param,
    OperationType operationType)
```

Handler for getting battery voltage.

**Parameters**

<i>param</i>	Empty string expected
<i>operationType</i>	GET

**Returns**

Vector of Frames containing:

- Success: Battery voltage in Volts
- Error: Error message

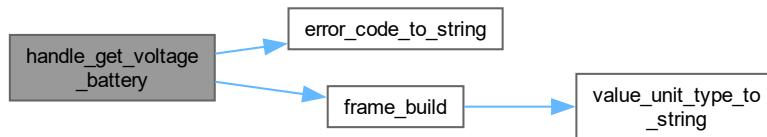
**Note****KBST;0;GET;2;2;;TSBK**

This command is used to retrieve the battery voltage

**Command** Command ID: 2.2

Definition at line 64 of file [power\\_commands.cpp](#).

Here is the call graph for this function:



### 6.6.2.3 handle\_get\_voltage\_5v()

```
std::vector< Frame > handle_get_voltage_5v (
    const std::string & param,
    OperationType operationType)
```

Handler for getting 5V rail voltage.

**Parameters**

<i>param</i>	Empty string expected
<i>operationType</i>	GET

**Returns**

Vector of Frames containing:

- Success: 5V rail voltage in Volts
- Error: Error message

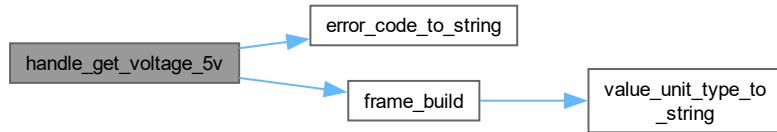
**Note****KBST;0;GET;2;3;;TSBK**

This command is used to retrieve the 5V rail voltage

**Command** Command ID: 2.3

Definition at line 98 of file [power\\_commands.cpp](#).

Here is the call graph for this function:



#### 6.6.2.4 handle\_get\_current\_charge\_usb()

```
std::vector< Frame > handle_get_current_charge_usb (
    const std::string & param,
    OperationType operationType)
```

Handler for getting USB charge current.

##### Parameters

<code>param</code>	Empty string expected
<code>operationType</code>	GET

##### Returns

Vector of Frames containing:

- Success: USB charge current in millamps
- Error: Error message

##### Note

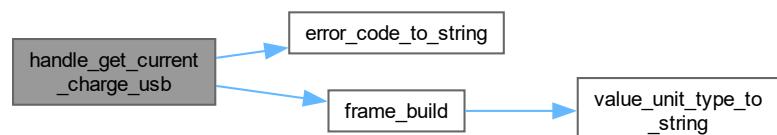
**KBST;0;GET;2;4;;TSBK**

This command is used to retrieve the USB charge current

**Command** Command ID: 2.4

Definition at line 132 of file [power\\_commands.cpp](#).

Here is the call graph for this function:



### 6.6.2.5 handle\_get\_current\_charge\_solar()

```
std::vector< Frame > handle_get_current_charge_solar (
    const std::string & param,
    OperationType operationType)
```

Handler for getting solar panel charge current.

#### Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	GET

#### Returns

Vector of Frames containing:

- Success: Solar charge current in millamps
- Error: Error message

#### Note

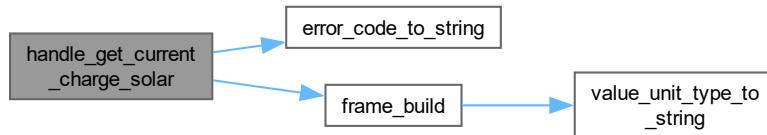
**KBST;0;GET;2;5;;TSBK**

This command is used to retrieve the solar panel charge current

**Command** Command ID: 2.5

Definition at line 166 of file [power\\_commands.cpp](#).

Here is the call graph for this function:



### 6.6.2.6 handle\_get\_current\_charge\_total()

```
std::vector< Frame > handle_get_current_charge_total (
    const std::string & param,
    OperationType operationType)
```

Handler for getting total charge current.

## Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	GET

## Returns

Vector of Frames containing:

- Success: Total charge current (USB + Solar) in millamps
- Error: Error message

## Note

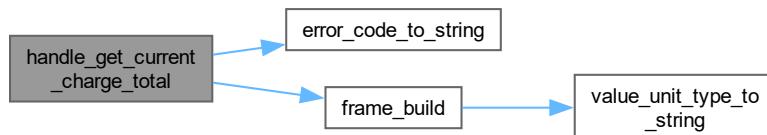
**KBST;0;GET;2;6;;TSBK**

This command is used to retrieve the total charge current

**Command** Command ID: 2.6

Definition at line 200 of file [power\\_commands.cpp](#).

Here is the call graph for this function:



### 6.6.2.7 handle\_get\_current\_draw()

```
std::vector< Frame > handle_get_current_draw (
    const std::string & param,
    OperationType operationType)
```

Handler for getting system current draw.

## Parameters

<i>param</i>	Empty string expected
<i>operationType</i>	GET

**Returns**

Vector of Frames containing:

- Success: System current consumption in millamps
- Error: Error message

**Note**

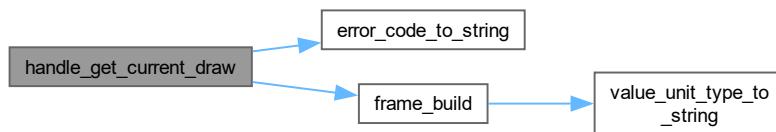
**KBST;0;GET;2;7;;TSBK**

This command is used to retrieve the system current draw

**Command** Command ID: 2.7

Definition at line 234 of file [power\\_commands.cpp](#).

Here is the call graph for this function:



## 6.7 Sensor Commands

Commands for reading and configuring sensors.

**Functions**

- `std::vector< Frame > handle_get_sensor_data (const std::string &param, OperationType operationType)`  
*Handler for reading sensor data.*
- `std::vector< Frame > handle_sensor_config (const std::string &param, OperationType operationType)`  
*Handler for configuring sensors.*
- `std::vector< Frame > handle_get_sensor_list (const std::string &param, OperationType operationType)`  
*Handler for listing available sensors.*

### 6.7.1 Detailed Description

Commands for reading and configuring sensors.

### 6.7.2 Function Documentation

#### 6.7.2.1 handle\_get\_sensor\_data()

```
std::vector< Frame > handle_get_sensor_data (
    const std::string & param,
    OperationType operationType)
```

Handler for reading sensor data.

## Parameters

<i>param</i>	String in format "sensor_type[-data_type]" where: <ul style="list-style-type: none"> <li>• sensor_type: "light", "environment", "magnetometer", "imu"</li> <li>• data_type (optional): specific data type for the sensor <ul style="list-style-type: none"> <li>– For light: "light_level"</li> <li>– For environment: "temperature", "pressure", "humidity"</li> <li>– For magnetometer: "mag_field_x", "mag_field_y", "mag_field_z"</li> <li>– For IMU: "gyro_x", "gyro_y", "gyro_z", "accel_x", "accel_y", "accel_z"</li> </ul> </li> </ul>
<i>operationType</i>	GET

## Returns

Vector of Frames containing:

- Success: Sensor data value(s)
- Error: Error message

## Note

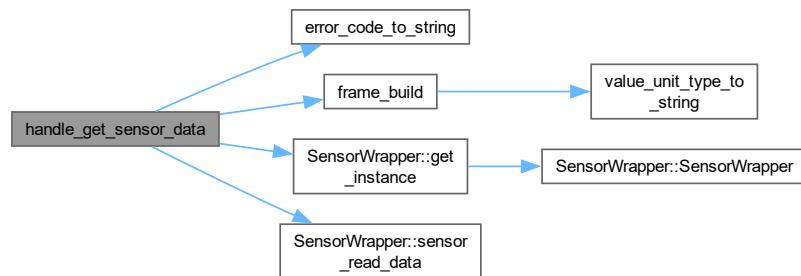
**KBST;0;GET;3;0;light-light\_level;TSBK**

This command is used to read data from sensors

**Command** Command ID: 3.0

Definition at line 36 of file [sensor\\_commands.cpp](#).

Here is the call graph for this function:



### 6.7.2.2 handle\_sensor\_config()

```
std::vector< Frame > handle_sensor_config (
    const std::string & param,
    OperationType operationType)
```

Handler for configuring sensors.

### Parameters

<i>param</i>	String in format "sensor_type;key1:value1 key2:value2 ..." <ul style="list-style-type: none"><li>• sensor_type: "light", "environment", "magnetometer", "imu"</li><li>• key-value pairs for configuration parameters</li></ul>
<i>operationType</i>	SET

### Returns

Vector of Frames containing:

- Success: Success message
- Error: Error message

### Note

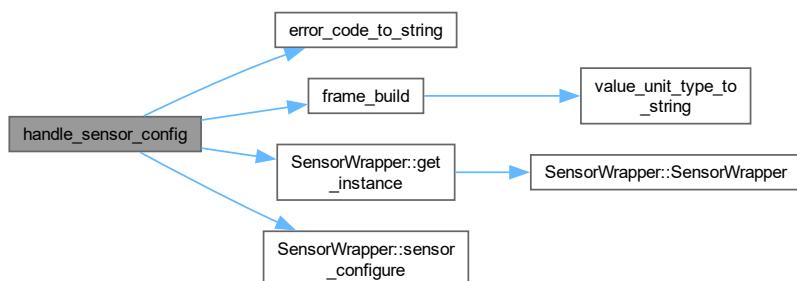
**KBST;0;SET;3;1;light;measurement\_mode:continuously\_high\_resolution;TSBK**

This command is used to configure sensors

**Command** Command ID: 3.1

Definition at line 236 of file [sensor\\_commands.cpp](#).

Here is the call graph for this function:



### 6.7.2.3 handle\_get\_sensor\_list()

```
std::vector< Frame > handle_get_sensor_list (
    const std::string & param,
    OperationType operationType)
```

Handler for listing available sensors.

**Parameters**

<i>param</i>	Empty string or optional filter criteria
<i>operationType</i>	GET

**Returns**

Vector of Frames containing:

- Success: List of available sensors
- Error: Error message

**Note**

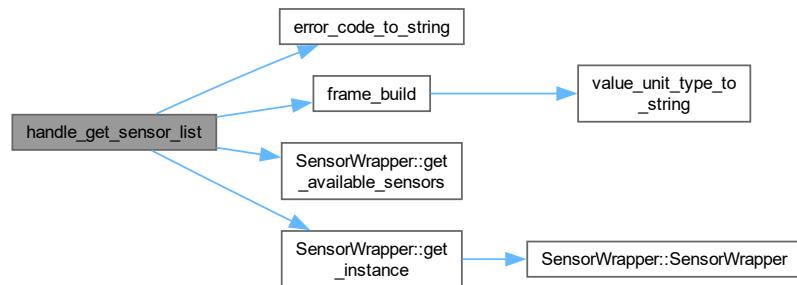
**KBST;0;GET;4;2;;TSBK** (lists all sensors)

This command is used to get a list of available sensors

**Command** Command ID: 4.2

Definition at line 320 of file [sensor\\_commands.cpp](#).

Here is the call graph for this function:



## 6.8 Storage Commands

Commands for interacting with the SD card storage.

**Functions**

- `std::vector< Frame > handle_list_files (const std::string &param, OperationType operationType)`  
*Handles the list files command.*
- `std::vector< Frame > handle_mount (const std::string &param, OperationType operationType)`  
*Handles the SD card mount/unmount command.*

## 6.8.1 Detailed Description

Commands for interacting with the SD card storage.

## 6.8.2 Function Documentation

### 6.8.2.1 handle\_list\_files()

```
std::vector< Frame > handle_list_files (
    const std::string & param,
    OperationType operationType)
```

Handles the list files command.

This function lists the files in the root directory of the SD card and sends the filename and size of each file to the ground station.

#### Parameters

<i>param</i>	Unused.
<i>operationType</i>	The operation type (must be GET).

#### Returns

A vector of Frames indicating the result of the operation.

- Success: Frame with "File listing complete" message.
- Error: Frame with error message (e.g., "Could not open directory").

#### Note

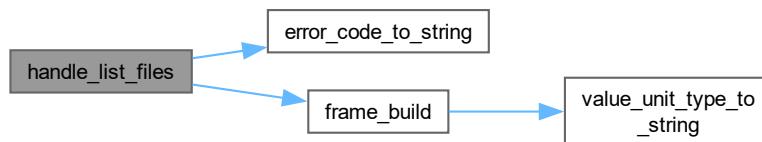
**KBST;0;GET;6;0;;TSBK**

This command lists the files and their sizes in the root directory of the SD card.

**Command** Command ID: 6.0

Definition at line 37 of file [storage\\_commands.cpp](#).

Here is the call graph for this function:



### 6.8.2.2 handle\_mount()

```
std::vector< Frame > handle_mount (
    const std::string & param,
    OperationType operationType)
```

Handles the SD card mount/unmount command.

This function mounts or unmounts the SD card.

### Parameters

<code>param</code>	"0" to unmount, "1" to mount.
<code>operationType</code>	The operation type (must be SET).

### Returns

A vector of Frames indicating the result of the operation.

- Success: [Frame](#) with "SD card mounted" or "SD card unmounted" message.
- Error: [Frame](#) with error message (e.g., "Invalid parameter", "Mount failed", "Unmount failed").

### Note

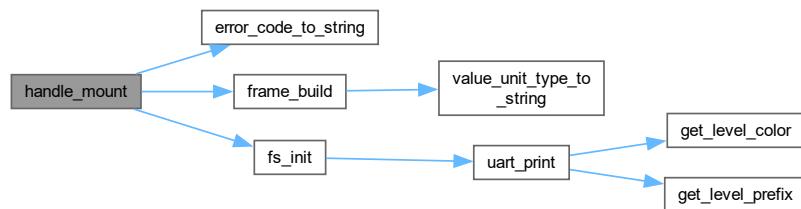
**KBST;0;SET;6;4;[0|1];TSBK**

Example: **KBST;0;SET;6;4;1;TSBK** - Mounts the SD card.

**Command** Command ID: 6.4

Definition at line 123 of file [storage\\_commands.cpp](#).

Here is the call graph for this function:



## 6.9 Frame Handling

Functions for encoding, decoding and building communication frames.

### Functions

- `std::string frame_encode (const Frame &frame)`  
*Encodes a [Frame](#) instance into a string.*
- `Frame frame_decode (const std::string &data)`  
*Decodes a string into a [Frame](#) instance.*
- `void frame_process (const std::string &data, Interface interface)`  
*Executes a command based on the command key and the parameter.*
- `Frame frame_build (OperationType operation, uint8_t group, uint8_t command, const std::string &value, const ValueUnit unitType)`  
*Builds a [Frame](#) instance based on the execution result, group, command, value, and unit.*

## 6.9.1 Detailed Description

Functions for encoding, decoding and building communication frames.

## 6.9.2 Function Documentation

### 6.9.2.1 frame\_encode()

```
std::string frame_encode (
    const Frame & frame)
```

Encodes a [Frame](#) instance into a string.

#### Parameters

<i>frame</i>	The <a href="#">Frame</a> instance to encode.
--------------	---

#### Returns

The [Frame](#) encoded as a string.

The encoded string includes the frame direction, operation type, group, command, value, and unit, all delimited by the DELIMITER character. The string is encapsulated by FRAME\_BEGIN and FRAME\_END.

```
Frame myFrame;
myFrame.header = FRAME_BEGIN;
myFrame.direction = 0;
myFrame.operationType = OperationType::GET;
myFrame.group = 1;
myFrame.command = 1;
myFrame.value = "";
myFrame.unit = "";
myFrame.footer = FRAME_END;

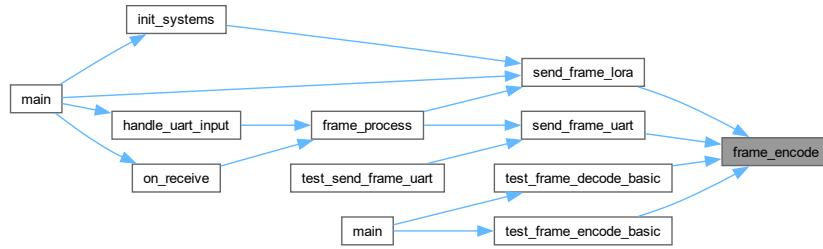
std::string encoded = frame_encode(myFrame);
// encoded will be "KBST;0;GET;1;1;;TSBK"
```

Definition at line 37 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.9.2.2 frame\_decode()

```
Frame frame_decode (
    const std::string & data)
```

Decodes a string into a [Frame](#) instance.

#### Parameters

<i>encodedFrame</i>	The string to decode.
---------------------	-----------------------

#### Returns

The [Frame](#) instance decoded from the string.

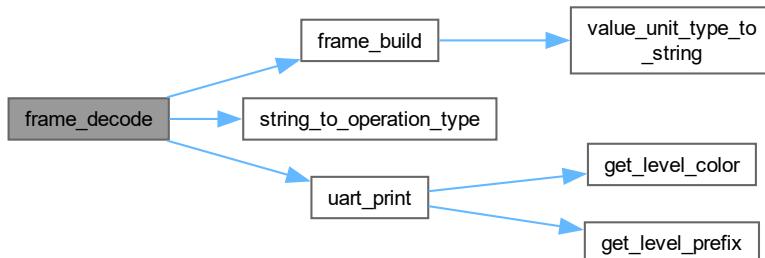
#### Exceptions

<i>std::runtime_error</i>	if the frame is invalid.
---------------------------	--------------------------

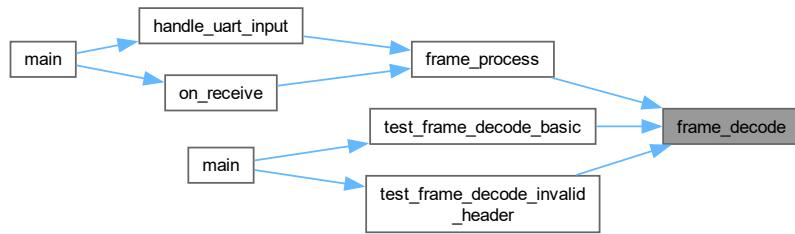
The decoded string is expected to be in the format: FRAME\_BEGIN;direction;operationType;group;command;value;unit;FRAME-END

Definition at line 62 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.9.2.3 frame\_process()

```
void frame_process (
    const std::string & data,
    Interface interface)
```

Executes a command based on the command key and the parameter.

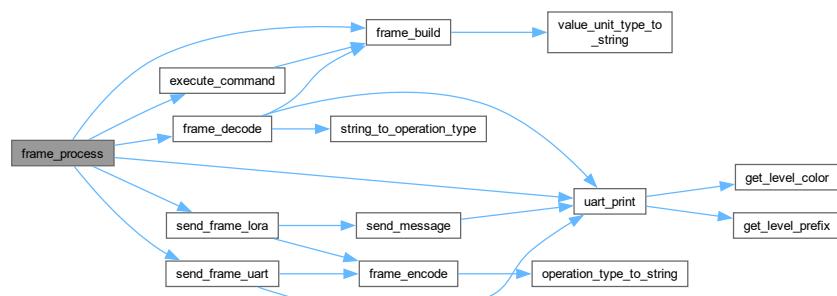
#### Parameters

<code>data</code>	The Frame data in string format.
-------------------	----------------------------------

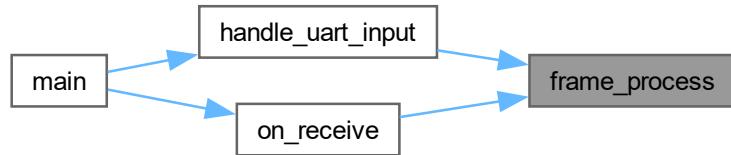
Decodes the frame data, extracts the command key, and executes the corresponding command. Sends the response frame. If an error occurs, an error frame is built and sent.

Definition at line 117 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.9.2.4 frame\_build()

```
Frame frame_build (
    OperationType operation,
    uint8_t group,
    uint8_t command,
    const std::string & value,
    const ValueUnit unitType)
```

Builds a [Frame](#) instance based on the execution result, group, command, value, and unit.

##### Parameters

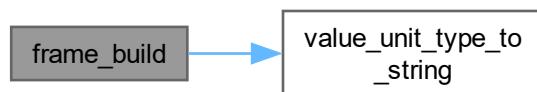
<i>result</i>	The execution result.
<i>group</i>	The group ID.
<i>command</i>	The command ID within the group.
<i>value</i>	The payload value.
<i>unit</i>	The unit of measurement for the payload value.

##### Returns

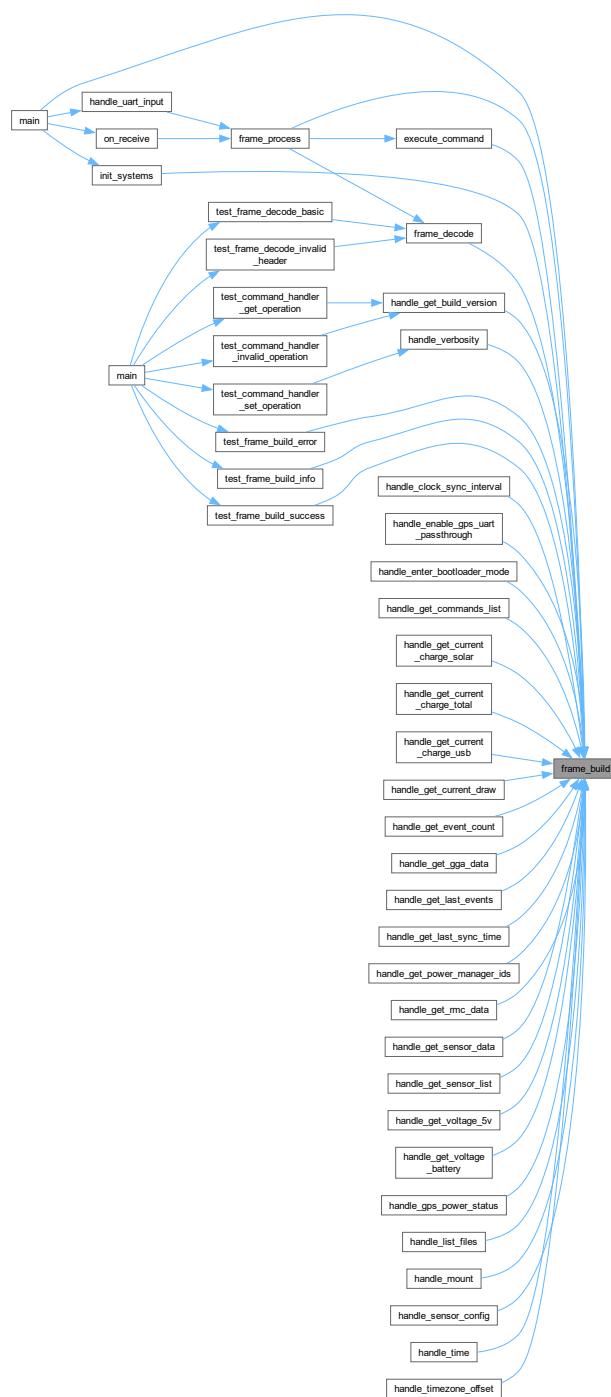
The [Frame](#) instance.

Definition at line 158 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 6.10 Event Manager

Classes and functions for managing event logging.

## Files

- file `event_manager.cpp`  
*Implements the event management system for the Kabisat firmware.*

## Classes

- class `EventLog`  
*Represents a single event log entry.*
- class `EventManager`  
*Manages the event logging system.*
- class `EventManagerImpl`  
*Implementation of the `EventManager` class.*
- class `EventEmitter`  
*Provides a static method for emitting events.*

## Enumerations

- enum class `EventGroup` : `uint8_t`{  
  `EventGroup::SYSTEM` = 0x00 , `EventGroup::POWER` = 0x01 , `EventGroup::COMMS` = 0x02 ,  
  `EventGroup::GPS` = 0x03 ,  
  `EventGroup::CLOCK` = 0x04 }  
*Represents the group to which an event belongs.*
- enum class `SystemEvent` : `uint8_t`{  
  `SystemEvent::BOOT` = 0x01 , `SystemEvent::SHUTDOWN` = 0x02 , `SystemEvent::WATCHDOG_RESET` = 0x03 , `SystemEvent::CORE1_START` = 0x04 ,  
  `SystemEvent::CORE1_STOP` = 0x05 }  
*Represents specific system events.*
- enum class `PowerEvent` : `uint8_t`{  
  `PowerEvent::LOW_BATTERY` = 0x01 , `PowerEvent::OVERCHARGE` = 0x02 , `PowerEvent::POWER_FALLING` = 0x03 , `PowerEvent::POWER_NORMAL` = 0x04 ,  
  `PowerEvent::SOLAR_ACTIVE` = 0x05 , `PowerEvent::SOLAR_INACTIVE` = 0x06 , `PowerEvent::USB_CONNECTED` = 0x07 , `PowerEvent::USB_DISCONNECTED` = 0x08 }  
*Represents specific power-related events.*
- enum class `CommsEvent` : `uint8_t`{  
  `CommsEvent::RADIO_INIT` = 0x01 , `CommsEvent::RADIO_ERROR` = 0x02 , `CommsEvent::MSG RECEIVED` = 0x03 , `CommsEvent::MSG SENT` = 0x04 ,  
  `CommsEvent::UART_ERROR` = 0x06 }  
*Represents specific communication-related events.*
- enum class `GPSEvent` : `uint8_t`{  
  `GPSEvent::LOCK` = 0x01 , `GPSEvent::LOST` = 0x02 , `GPSEvent::ERROR` = 0x03 , `GPSEvent::POWER_ON` = 0x04 ,  
  `GPSEvent::POWER_OFF` = 0x05 , `GPSEvent::DATA_READY` = 0x06 , `GPSEvent::PASS_THROUGH_START` = 0x07 , `GPSEvent::PASS_THROUGH_END` = 0x08 }  
*Represents specific GPS-related events.*
- enum class `ClockEvent` : `uint8_t` { `ClockEvent::CHANGED` = 0x01 , `ClockEvent::GPS_SYNC` = 0x02 ,  
  `ClockEvent::GPS_SYNC_DATA_NOT_READY` = 0x03 }  
*Represents specific clock-related events.*

## Functions

- class `EventLog __attribute__ ((packed))`
- void `EventManager::log_event (uint8_t group, uint8_t event)`  
*Logs an event.*
- const `EventLog & EventManager::get_event (size_t index) const`  
*Retrieves an event from the event buffer.*

## Variables

- volatile `uint16_t eventId = 0`  
*Global event log ID counter.*
- `DS3231 systemClock`  
*External declaration of the system clock.*
- `EventManagerImpl eventManager`  
*Global instance of the `EventManager` implementation.*
- class `EventManager __attribute__`
- `EventManagerImpl eventManager`  
*Global instance of the `EventManagerImpl` class.*

### 6.10.1 Detailed Description

Classes and functions for managing event logging.

### 6.10.2 Enumeration Type Documentation

#### 6.10.2.1 EventGroup

```
enum class EventGroup : uint8_t [strong]
```

Represents the group to which an event belongs.

##### Enumerator

<code>SYSTEM</code>	System events.
<code>POWER</code>	Power-related events.
<code>COMMS</code>	Communication-related events.
<code>GPS</code>	GPS-related events.
<code>CLOCK</code>	Clock-related events.

Definition at line 28 of file `event_manager.h`.

#### 6.10.2.2 SystemEvent

```
enum class SystemEvent : uint8_t [strong]
```

Represents specific system events.

## Enumerator

BOOT	System boot event.
SHUTDOWN	System shutdown event.
WATCHDOG_RESET	Watchdog reset event.
CORE1_START	Core 1 start event.
CORE1_STOP	Core 1 stop event.

Definition at line 45 of file [event\\_manager.h](#).

**6.10.2.3 PowerEvent**

```
enum class PowerEvent : uint8_t [strong]
```

Represents specific power-related events.

## Enumerator

LOW_BATTERY	Low battery event.
OVERCHARGE	Overcharge event.
POWER_FALLING	Power falling event.
POWER_NORMAL	Power normal event.
SOLAR_ACTIVE	Solar charging active event.
SOLAR_INACTIVE	Solar charging inactive event.
USB_CONNECTED	USB connected event.
USB_DISCONNECTED	USB disconnected event.

Definition at line 62 of file [event\\_manager.h](#).

**6.10.2.4 CommsEvent**

```
enum class CommsEvent : uint8_t [strong]
```

Represents specific communication-related events.

## Enumerator

RADIO_INIT	Radio initialization event.
RADIO_ERROR	Radio error event.
MSG RECEIVED	Message received event.
MSG SENT	Message sent event.
UART_ERROR	UART error event.

Definition at line 85 of file [event\\_manager.h](#).

**6.10.2.5 GPSEvent**

```
enum class GPSEvent : uint8_t [strong]
```

Represents specific GPS-related events.

**Enumerator**

LOCK	GPS lock acquired event.
LOST	GPS lock lost event.
ERROR	GPS error event.
POWER_ON	GPS power on event.
POWER_OFF	GPS power off event.
DATA_READY	GPS data ready event.
PASS_THROUGH_START	GPS pass-through start event.
PASS_THROUGH_END	GPS pass-through end event.

Definition at line 102 of file [event\\_manager.h](#).

**6.10.2.6 ClockEvent**

```
enum class ClockEvent : uint8_t [strong]
```

Represents specific clock-related events.

**Enumerator**

CHANGED	Clock changed event.
GPS_SYNC	Clock synchronized with GPS event.
GPS_SYNC_DATA_NOT_READY	Sync interval but data not ready.

Definition at line 126 of file [event\\_manager.h](#).

**6.10.3 Function Documentation****6.10.3.1 \_\_attribute\_\_()**

```
class EventLog __attribute__ (
    packed) }
```

**6.10.3.2 log\_event()**

```
void EventManager::log_event (
    uint8_t group,
    uint8_t event)
```

Logs an event.

Logs an event to the event buffer.

**Parameters**

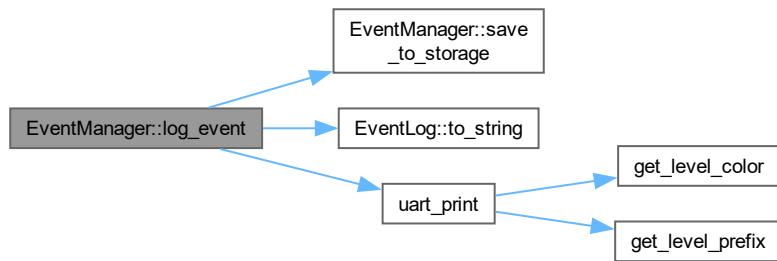
group	The event group.
event	The event identifier.

<i>group</i>	The event group.
<i>event</i>	The event ID.

Logs the event with a timestamp, group, and event ID. Prints the event to the UART, and saves the event to storage if the buffer is full or if it's a power-related event.

Definition at line 45 of file [event\\_manager.cpp](#).

Here is the call graph for this function:



### 6.10.3.3 `get_event()`

```
const EventLog & EventManager::get_event (
    size_t index) const
```

Retrieves an event from the event buffer.

#### Parameters

<i>index</i>	The index of the event to retrieve.
--------------	-------------------------------------

#### Returns

A const reference to the [EventLog](#) at the specified index.

#### Parameters

<i>index</i>	The index of the event to retrieve.
--------------	-------------------------------------

#### Returns

A const reference to the [EventLog](#) at the specified index. Returns an empty event if the index is out of bounds.

Definition at line 83 of file [event\\_manager.cpp](#).

## 6.10.4 Variable Documentation

### 6.10.4.1 eventLogId

```
volatile uint16_t eventLogId = 0
```

Global event log ID counter.

Definition at line 21 of file [event\\_manager.cpp](#).

### 6.10.4.2 systemClock

```
DS3231 systemClock [extern]
```

External declaration of the system clock.

### 6.10.4.3 eventManager [1/2]

```
EventManagerImpl eventManager
```

Global instance of the [EventManager](#) implementation.

Global instance of the [EventManagerImpl](#) class.

Definition at line 33 of file [event\\_manager.cpp](#).

### 6.10.4.4 \_\_attribute\_\_

```
class EventManager __attribute__
```

### 6.10.4.5 eventManager [2/2]

```
EventManagerImpl eventManager [extern]
```

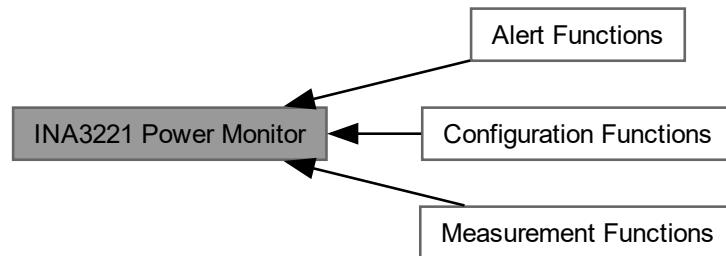
Global instance of the [EventManagerImpl](#) class.

Global instance of the [EventManagerImpl](#) class.

Definition at line 33 of file [event\\_manager.cpp](#).

## 6.11 INA3221 Power Monitor

Collaboration diagram for INA3221 Power Monitor:



### Topics

- Configuration Functions
- Measurement Functions
- Alert Functions

#### 6.11.1 Detailed Description

#### 6.11.2 Configuration Functions

Collaboration diagram for Configuration Functions:



### Functions

- `INA3221::INA3221 (ina3221_addr_t addr, i2c_inst_t *i2c)`  
*Constructor for `INA3221` class.*
- `bool INA3221::begin ()`  
*Initialize the `INA3221` device.*
- `void INA3221::reset ()`  
*Reset the `INA3221` to default settings.*
- `uint16_t INA3221::get_manufacturer_id ()`

- `uint16_t INA3221::get_die_id ()`  
*Get the manufacturer ID of the device.*
- `uint16_t INA3221::read_register (ina3221_reg_t reg)`  
*Read a register from the device.*
- `void INA3221::set_mode_power_down ()`  
*Set device to power-down mode.*
- `void INA3221::set_mode_continuous ()`  
*Set device to continuous measurement mode.*
- `void INA3221::set_mode_triggered ()`  
*Set device to triggered measurement mode.*
- `void INA3221::set_shunt_measurement_enable ()`  
*Enable shunt voltage measurements.*
- `void INA3221::set_shunt_measurement_disable ()`  
*Disable shunt voltage measurements.*
- `void INA3221::set_bus_measurement_enable ()`  
*Enable bus voltage measurements.*
- `void INA3221::set_bus_measurement_disable ()`  
*Disable bus voltage measurements.*
- `void INA3221::set_averaging_mode (ina3221_avg_mode_t mode)`  
*Set the averaging mode for measurements.*
- `void INA3221::set_bus_conversion_time (ina3221_conv_time_t convTime)`  
*Set bus voltage conversion time.*
- `void INA3221::set_shunt_conversion_time (ina3221_conv_time_t convTime)`  
*Set shunt voltage conversion time.*

### 6.11.2.1 Detailed Description

Functions for configuring the [INA3221](#) device

### 6.11.2.2 Function Documentation

#### 6.11.2.2.1 [INA3221\(\)](#)

```
INA3221::INA3221 (
    ina3221_addr_t addr,
    i2c_inst_t * i2c)
```

Constructor for [INA3221](#) class.

##### Parameters

<code>addr</code>	I2C address of the device
<code>i2c</code>	Pointer to I2C instance

Definition at line 46 of file [INA3221.cpp](#).

### 6.11.2.2.2 begin()

```
bool INA3221::begin ()
```

Initialize the [INA3221](#) device.

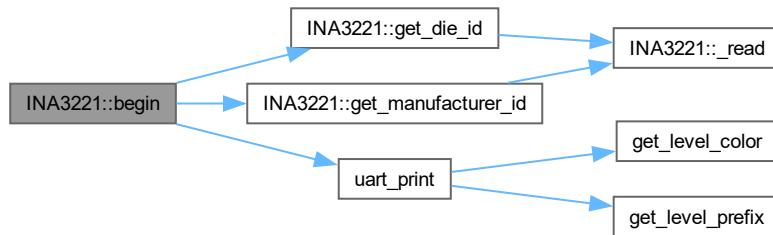
#### Returns

true if initialization successful, false otherwise

Sets up shunt resistors, filter resistors, and verifies device IDs

Definition at line 56 of file [INA3221.cpp](#).

Here is the call graph for this function:



### 6.11.2.2.3 reset()

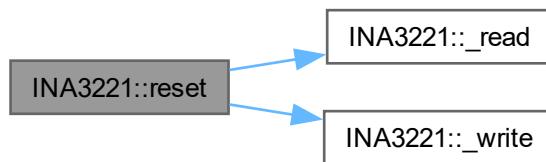
```
void INA3221::reset ()
```

Reset the [INA3221](#) to default settings.

Performs a software reset of the device by setting the reset bit

Definition at line 90 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.2.2.4 `get_manufacturer_id()`

```
uint16_t INA3221::get_manufacturer_id ()
```

Get the manufacturer ID of the device.

##### Returns

16-bit manufacturer ID (should be 0x5449)

Definition at line 104 of file [INA3221.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.11.2.2.5 `get_die_id()`

```
uint16_t INA3221::get_die_id ()
```

Get the die ID of the device.

**Returns**

16-bit die ID (should be 0x3220)

Definition at line 116 of file [INA3221.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 6.11.2.2.6 `read_register()`

```
uint16_t INA3221::read_register (
    ina3221_reg_t reg)
```

Read a register from the device.

**Parameters**

<i>reg</i>	Register address to read
------------	--------------------------

**Returns**

16-bit value read from the register

Definition at line 129 of file [INA3221.cpp](#).

Here is the call graph for this function:



### 6.11.2.2.7 set\_mode\_power\_down()

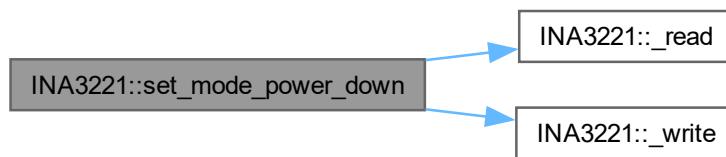
```
void INA3221::set_mode_power_down ()
```

Set device to power-down mode.

Disables bus voltage and continuous measurements

Definition at line 143 of file [INA3221.cpp](#).

Here is the call graph for this function:



### 6.11.2.2.8 set\_mode\_continuous()

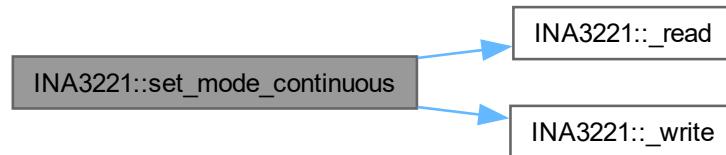
```
void INA3221::set_mode_continuous ()
```

Set device to continuous measurement mode.

Enables continuous measurement of bus voltage and shunt voltage

Definition at line 158 of file [INA3221.cpp](#).

Here is the call graph for this function:



### 6.11.2.2.9 set\_mode\_triggered()

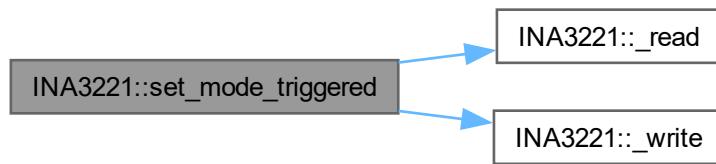
```
void INA3221::set_mode_triggered ()
```

Set device to triggered measurement mode.

Disables continuous measurements, requiring manual triggers

Definition at line 172 of file [INA3221.cpp](#).

Here is the call graph for this function:



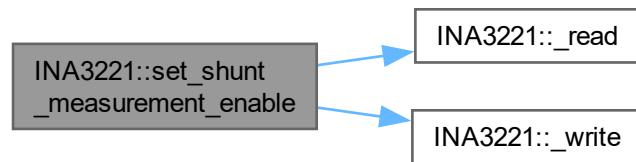
### 6.11.2.2.10 set\_shunt\_measurement\_enable()

```
void INA3221::set_shunt_measurement_enable ()
```

Enable shunt voltage measurements.

Definition at line 185 of file [INA3221.cpp](#).

Here is the call graph for this function:



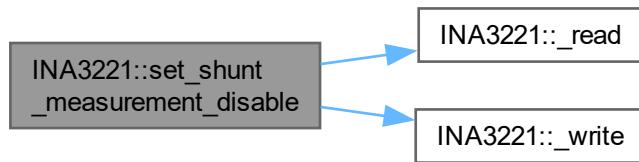
### 6.11.2.2.11 set\_shunt\_measurement\_disable()

```
void INA3221::set_shunt_measurement_disable ()
```

Disable shunt voltage measurements.

Definition at line 198 of file [INA3221.cpp](#).

Here is the call graph for this function:



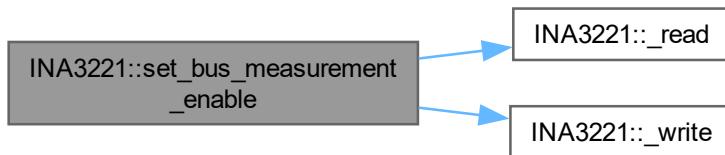
### 6.11.2.2.12 set\_bus\_measurement\_enable()

```
void INA3221::set_bus_measurement_enable ()
```

Enable bus voltage measurements.

Definition at line 211 of file [INA3221.cpp](#).

Here is the call graph for this function:



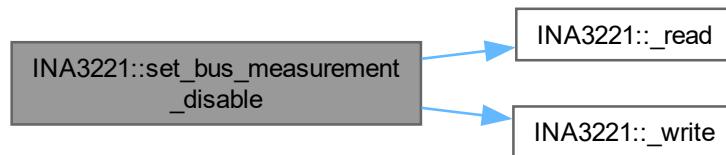
#### 6.11.2.2.13 set\_bus\_measurement\_disable()

```
void INA3221::set_bus_measurement_disable ()
```

Disable bus voltage measurements.

Definition at line 224 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.2.2.14 set\_averaging\_mode()

```
void INA3221::set_averaging_mode (
    ina3221_avg_mode_t mode)
```

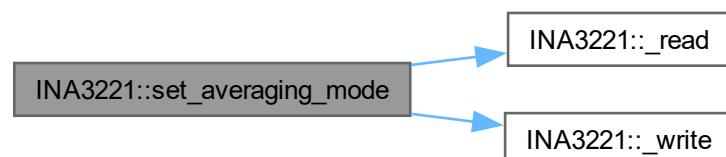
Set the averaging mode for measurements.

##### Parameters

<code>mode</code>	Number of samples to average
-------------------	------------------------------

Definition at line 238 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.2.2.15 set\_bus\_conversion\_time()

```
void INA3221::set_bus_conversion_time (
    ina3221_conv_time_t convTime)
```

Set bus voltage conversion time.

**Parameters**

<i>convTime</i>	Conversion time setting
-----------------	-------------------------

Definition at line 252 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.2.2.16 `set_shunt_conversion_time()`

```
void INA3221::set_shunt_conversion_time (
    ina3221_conv_time_t convTime)
```

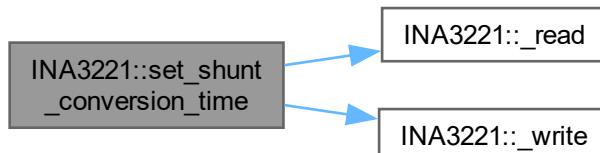
Set shunt voltage conversion time.

**Parameters**

<i>convTime</i>	Conversion time setting
-----------------	-------------------------

Definition at line 266 of file [INA3221.cpp](#).

Here is the call graph for this function:



### 6.11.3 Measurement Functions

Collaboration diagram for Measurement Functions:



#### Functions

- int32\_t `INA3221::get_shunt_voltage (ina3221_ch_t channel)`  
*Get shunt voltage for a specific channel.*
- float `INA3221::get_current_ma (ina3221_ch_t channel)`  
*Get current for a specific channel.*
- float `INA3221::get_voltage (ina3221_ch_t channel)`  
*Get bus voltage for a specific channel.*

#### 6.11.3.1 Detailed Description

Functions for reading voltage, current and power measurements

#### 6.11.3.2 Function Documentation

##### 6.11.3.2.1 `get_shunt_voltage()`

```
int32_t INA3221::get_shunt_voltage (
    ina3221_ch_t channel)
```

Get shunt voltage for a specific channel.

##### Parameters

<code>channel</code>	Channel number (1-3)
----------------------	----------------------

**Returns**

Shunt voltage in microvolts ( $\mu$ V)

Definition at line 282 of file [INA3221.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.11.3.2.2 get\_current\_ma()

```
float INA3221::get_current_ma (
    ina3221_ch_t channel)
```

Get current for a specific channel.

**Parameters**

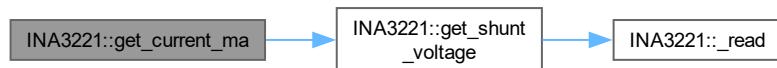
<i>channel</i>	Channel number (1-3)
----------------	----------------------

**Returns**

Current in millamps (mA)

Definition at line 314 of file [INA3221.cpp](#).

Here is the call graph for this function:



### 6.11.3.2.3 `get_voltage()`

```
float INA3221::get_voltage (
    ina3221_ch_t channel)
```

Get bus voltage for a specific channel.

#### Parameters

<code>channel</code>	Channel number (1-3)
----------------------	----------------------

#### Returns

Voltage in volts (V)

Definition at line 330 of file [INA3221.cpp](#).

Here is the call graph for this function:



## 6.11.4 Alert Functions

Collaboration diagram for Alert Functions:



### Functions

- void [INA3221::set\\_warn\\_alert\\_limit](#) (ina3221\_ch\_t channel, float voltage\_v)  
*Set warning alert voltage threshold for a channel.*
- void [INA3221::set\\_crit\\_alert\\_limit](#) (ina3221\_ch\_t channel, float voltage\_v)  
*Set critical alert voltage threshold for a channel.*
- void [INA3221::set\\_power\\_valid\\_limit](#) (float voltage\_upper\_v, float voltage\_lower\_v)  
*Set power valid voltage range.*

- void `INA3221::enable_alerts ()`  
*Enable all alert functions.*
- bool `INA3221::get_warn_alert (ina3221_ch_t channel)`  
*Get warning alert status for a channel.*
- bool `INA3221::get_crit_alert (ina3221_ch_t channel)`  
*Get critical alert status for a channel.*
- bool `INA3221::get_power_valid_alert ()`  
*Get power valid alert status.*
- void `INA3221::set_alert_latch (bool enable)`  
*Set alert latch mode.*

#### 6.11.4.1 Detailed Description

Functions for configuring and reading alert conditions

#### 6.11.4.2 Function Documentation

##### 6.11.4.2.1 `set_warn_alert_limit()`

```
void INA3221::set_warn_alert_limit (
    ina3221_ch_t channel,
    float voltage_v)
```

Set warning alert voltage threshold for a channel.

###### Parameters

<i>channel</i>	Channel number (1-3)
<i>voltage_v</i>	Voltage threshold in volts

Definition at line 360 of file [INA3221.cpp](#).

Here is the call graph for this function:



##### 6.11.4.2.2 `set_crit_alert_limit()`

```
void INA3221::set_crit_alert_limit (
    ina3221_ch_t channel,
    float voltage_v)
```

Set critical alert voltage threshold for a channel.

#### Parameters

<i>channel</i>	Channel number (1-3)
<i>voltage_v</i>	Voltage threshold in volts

Definition at line 385 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.4.2.3 set\_power\_valid\_limit()

```
void INA3221::set_power_valid_limit (
    float voltage_upper_v,
    float voltage_lower_v)
```

Set power valid voltage range.

#### Parameters

<i>voltage_upper_v</i>	Upper voltage threshold in volts
<i>voltage_lower_v</i>	Lower voltage threshold in volts

Definition at line 410 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.4.2.4 enable\_alerts()

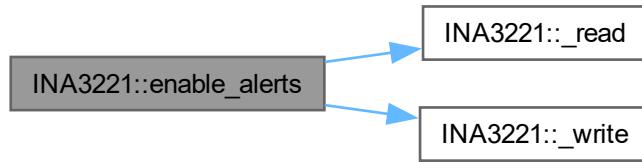
```
void INA3221::enable_alerts ()
```

Enable all alert functions.

Enables warning alerts, critical alerts, and power valid alerts for all channels

Definition at line 426 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.4.2.5 get\_warn\_alert()

```
bool INA3221::get_warn_alert (
    ina3221_ch_t channel)
```

Get warning alert status for a channel.

##### Parameters

<code>channel</code>	Channel number (1-3)
----------------------	----------------------

##### Returns

true if warning alert is active, false otherwise

Definition at line 448 of file [INA3221.cpp](#).

Here is the call graph for this function:



#### 6.11.4.2.6 `get_crit_alert()`

```
bool INA3221::get_crit_alert (
    ina3221_ch_t channel)
```

Get critical alert status for a channel.

**Parameters**

<i>channel</i>	Channel number (1-3)
----------------	----------------------

**Returns**

true if critical alert is active, false otherwise

Definition at line 467 of file [INA3221.cpp](#).

Here is the call graph for this function:

**6.11.4.2.7 get\_power\_valid\_alert()**

```
bool INA3221::get_power_valid_alert ()
```

Get power valid alert status.

**Returns**

true if power valid alert is active, false otherwise

Definition at line 485 of file [INA3221.cpp](#).

Here is the call graph for this function:

**6.11.4.2.8 set\_alert\_latch()**

```
void INA3221::set_alert_latch (
    bool enable)
```

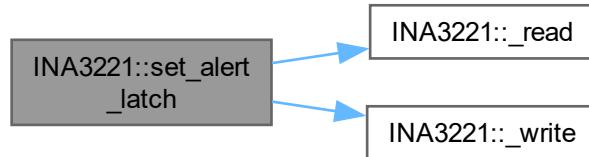
Set alert latch mode.

### Parameters

<code>enable</code>	true to enable alert latching, false for transparent alerts
---------------------	---

Definition at line 497 of file [INA3221.cpp](#).

Here is the call graph for this function:



## 6.12 Telemetry Manager

### Functions

- `bool telemetry_init ()`  
*Initialize the telemetry system.*
- `bool collect_telemetry ()`  
*Collect telemetry data from sensors and power subsystems.*
- `bool flush_telemetry ()`  
*Save buffered telemetry data to storage.*
- `bool is_telemetry_collection_time (uint32_t current_time, uint32_t &last_collection_time)`  
*Check if it's time to collect telemetry based on interval.*
- `bool is_telemetry_flush_time (uint32_t &collection_counter)`  
*Check if it's time to flush telemetry buffer based on count.*
- `uint32_t get_telemetry_sample_interval ()`  
*Get the current sample interval in milliseconds.*
- `void set_telemetry_sample_interval (uint32_t interval_ms)`  
*Set the telemetry sample interval.*
- `uint32_t get_telemetry_flush_threshold ()`  
*Get the number of records before flushing to storage.*
- `void set_telemetry_flush_threshold (uint32_t records)`  
*Set the number of records before flushing to storage.*

### 6.12.1 Detailed Description

### 6.12.2 Function Documentation

#### 6.12.2.1 `telemetry_init()`

```
bool telemetry_init ()
```

Initialize the telemetry system.

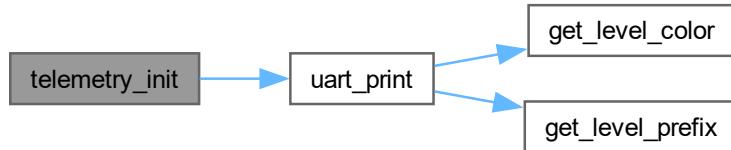
##### Returns

True if initialization was successful

Sets up the mutex for thread-safe buffer access and creates a telemetry CSV file with appropriate headers if it doesn't already exist

Definition at line 131 of file [telemetry\\_manager.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.12.2.2 collect\_telemetry()

```
bool collect_telemetry ()
```

Collect telemetry data from sensors and power subsystems.

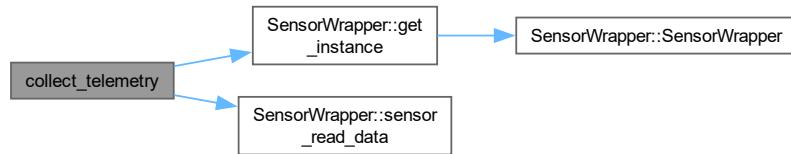
#### Returns

True if data was successfully collected

Reads data from power manager, sensors, and GPS and stores it in the telemetry buffer with proper mutex protection

Definition at line 160 of file [telemetry\\_manager.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 6.12.2.3 flush\_telemetry()

```
bool flush_telemetry ()
```

Save buffered telemetry data to storage.

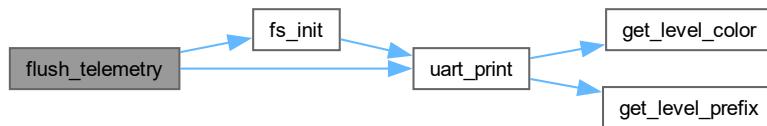
**Returns**

True if data was successfully saved

Writes all records from the telemetry buffer to the CSV file and clears the buffer after successful writing

Definition at line 228 of file [telemetry\\_manager.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**6.12.2.4 is\_telemetry\_collection\_time()**

```
bool is_telemetry_collection_time (
    uint32_t current_time,
    uint32_t & last_collection_time)
```

Check if it's time to collect telemetry based on interval.

**Parameters**

<code>current_time</code>	Current system time in milliseconds
<code>last_collection_time</code>	Previous collection time in milliseconds

**Returns**

True if collection interval has passed

Updates last\_collection\_time if the interval has passed

Definition at line 274 of file [telemetry\\_manager.cpp](#).

Here is the caller graph for this function:



### 6.12.2.5 is\_telemetry\_flush\_time()

```
bool is_telemetry_flush_time (
    uint32_t & collection_counter)
```

Check if it's time to flush telemetry buffer based on count.

**Parameters**

<i>collection_counter</i>	Current collection counter
---------------------------	----------------------------

**Returns**

True if flush threshold has been reached

Resets collection\_counter to zero if the threshold has been reached

Definition at line 282 of file [telemetry\\_manager.cpp](#).

Here is the caller graph for this function:



### 6.12.2.6 get\_telemetry\_sample\_interval()

```
uint32_t get_telemetry_sample_interval ()
```

Get the current sample interval in milliseconds.

#### Returns

Sample interval in milliseconds

Definition at line 290 of file [telemetry\\_manager.cpp](#).

### 6.12.2.7 set\_telemetry\_sample\_interval()

```
void set_telemetry_sample_interval (
    uint32_t interval_ms)
```

Set the telemetry sample interval.

#### Parameters

<i>interval_ms</i>	New interval in milliseconds
--------------------	------------------------------

Sets a minimum bound of 100ms to prevent excessive sampling

Definition at line 294 of file [telemetry\\_manager.cpp](#).

### 6.12.2.8 get\_telemetry\_flush\_threshold()

```
uint32_t get_telemetry_flush_threshold ()
```

Get the number of records before flushing to storage.

#### Returns

Number of records in flush threshold

Definition at line 300 of file [telemetry\\_manager.cpp](#).

### 6.12.2.9 set\_telemetry\_flush\_threshold()

```
void set_telemetry_flush_threshold (
    uint32_t records)
```

Set the number of records before flushing to storage.

#### Parameters

<i>records</i>	Number of records in flush threshold
----------------	--------------------------------------

Sets reasonable bounds (1-100) for the threshold value

Definition at line 304 of file [telemetry\\_manager.cpp](#).

# Chapter 7

## Class Documentation

### 7.1 BH1750 Class Reference

```
#include <BH1750.h>
```

#### Public Types

- enum class `Mode` : `uint8_t` {  
    `UNCONFIGURED_POWER_DOWN` = 0x00 , `POWER_ON` = 0x01 , `RESET` = 0x07 , `CONTINUOUS_HIGH_RES_MODE` = 0x10 ,  
    `CONTINUOUS_HIGH_RES_MODE_2` = 0x11 , `CONTINUOUS_LOW_RES_MODE` = 0x13 , `ONE_TIME_HIGH_RES_MODE` = 0x20 ,  
    `ONE_TIME_HIGH_RES_MODE_2` = 0x21 ,  
    `ONE_TIME_LOW_RES_MODE` = 0x23 }

#### Public Member Functions

- `BH1750` (`uint8_t` `addr`=0x23)
- bool `begin` (`Mode mode`=`Mode::CONTINUOUS_HIGH_RES_MODE`)
- void `configure` (`Mode mode`)
- float `get_light_level` ()

#### Private Member Functions

- void `write8` (`uint8_t data`)

#### Private Attributes

- `uint8_t _i2c_addr`

### 7.1.1 Detailed Description

Definition at line 12 of file `BH1750.h`.

### 7.1.2 Member Enumeration Documentation

#### 7.1.2.1 Mode

```
enum class BH1750::Mode : uint8_t [strong]
```

## Enumerator

UNCONFIGURED_POWER_DOWN	
POWER_ON	
RESET	
CONTINUOUS_HIGH_RES_MODE	
CONTINUOUS_HIGH_RES_MODE_2	
CONTINUOUS_LOW_RES_MODE	
ONE_TIME_HIGH_RES_MODE	
ONE_TIME_HIGH_RES_MODE_2	
ONE_TIME_LOW_RES_MODE	

Definition at line 15 of file [BH1750.h](#).

### 7.1.3 Constructor & Destructor Documentation

#### 7.1.3.1 BH1750()

```
BH1750::BH1750 (
    uint8_t addr = 0x23)
```

Definition at line 6 of file [BH1750.cpp](#).

### 7.1.4 Member Function Documentation

#### 7.1.4.1 begin()

```
bool BH1750::begin (
    Mode mode = Mode::CONTINUOUS_HIGH_RES_MODE)
```

Definition at line 8 of file [BH1750.cpp](#).

Here is the call graph for this function:



### 7.1.4.2 configure()

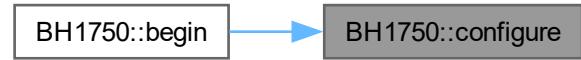
```
void BH1750::configure (
    Mode mode)
```

Definition at line 22 of file [BH1750.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.1.4.3 get\_light\_level()

```
float BH1750::get_light_level ()
```

Definition at line 40 of file [BH1750.cpp](#).

### 7.1.4.4 write8()

```
void BH1750::write8 (
    uint8_t data) [private]
```

Definition at line 49 of file [BH1750.cpp](#).

Here is the caller graph for this function:



## 7.1.5 Member Data Documentation

### 7.1.5.1 \_i2c\_addr

```
uint8_t BH1750::_i2c_addr [private]
```

Definition at line 34 of file [BH1750.h](#).

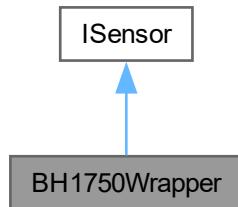
The documentation for this class was generated from the following files:

- lib/sensors/BH1750/[BH1750.h](#)
- lib/sensors/BH1750/[BH1750.cpp](#)

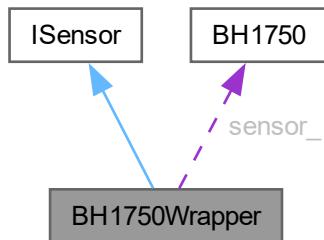
## 7.2 BH1750Wrapper Class Reference

```
#include <BH1750_WRAPPER.h>
```

Inheritance diagram for BH1750Wrapper:



Collaboration diagram for BH1750Wrapper:



## Public Member Functions

- `BH1750Wrapper ()`
- `int get_i2c_addr ()`
- `bool init () override`
- `float read_data (SensorDataTypelIdentifier type) override`
- `bool is_initialized () const override`
- `SensorType get_type () const override`
- `bool configure (const std::map< std::string, std::string > &config)`
- `uint8_t get_address () const override`

## Public Member Functions inherited from `ISensor`

- `virtual ~ISensor ()=default`

## Private Attributes

- `BH1750 sensor_`
- `bool initialized_ = false`

### 7.2.1 Detailed Description

Definition at line 9 of file `BH1750_WRAPPER.h`.

### 7.2.2 Constructor & Destructor Documentation

#### 7.2.2.1 `BH1750Wrapper()`

```
BH1750Wrapper::BH1750Wrapper ()
```

Definition at line 6 of file `BH1750_WRAPPER.cpp`.

### 7.2.3 Member Function Documentation

#### 7.2.3.1 `get_i2c_addr()`

```
int BH1750Wrapper::get_i2c_addr ()
```

#### 7.2.3.2 `init()`

```
bool BH1750Wrapper::init () [override], [virtual]
```

Implements `ISensor`.

Definition at line 10 of file `BH1750_WRAPPER.cpp`.

### 7.2.3.3 `read_data()`

```
float BH1750Wrapper::read_data (
    SensorDataTypeIdentifier type) [override], [virtual]
```

Implements [ISensor](#).

Definition at line 15 of file [BH1750\\_WRAPPER.cpp](#).

### 7.2.3.4 `is_initialized()`

```
bool BH1750Wrapper::is_initialized () const [override], [virtual]
```

Implements [ISensor](#).

Definition at line 22 of file [BH1750\\_WRAPPER.cpp](#).

### 7.2.3.5 `get_type()`

```
SensorType BH1750Wrapper::get_type () const [override], [virtual]
```

Implements [ISensor](#).

Definition at line 26 of file [BH1750\\_WRAPPER.cpp](#).

### 7.2.3.6 `configure()`

```
bool BH1750Wrapper::configure (
    const std::map< std::string, std::string > & config) [virtual]
```

Implements [ISensor](#).

Definition at line 30 of file [BH1750\\_WRAPPER.cpp](#).

### 7.2.3.7 `get_address()`

```
uint8_t BH1750Wrapper::get_address () const [inline], [override], [virtual]
```

Implements [ISensor](#).

Definition at line 24 of file [BH1750\\_WRAPPER.h](#).

## 7.2.4 Member Data Documentation

### 7.2.4.1 `sensor_`

```
BH1750 BH1750Wrapper::sensor_ [private]
```

Definition at line 11 of file [BH1750\\_WRAPPER.h](#).

### 7.2.4.2 initialized\_

```
bool BH1750Wrapper::initialized_ = false [private]
```

Definition at line 12 of file [BH1750\\_WRAPPER.h](#).

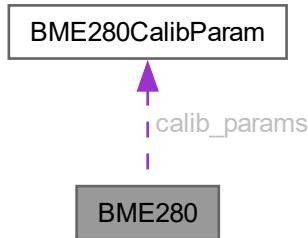
The documentation for this class was generated from the following files:

- lib/sensors/BH1750/[BH1750\\_WRAPPER.h](#)
- lib/sensors/BH1750/[BH1750\\_WRAPPER.cpp](#)

## 7.3 BME280 Class Reference

```
#include <BME280.h>
```

Collaboration diagram for BME280:



### Public Member Functions

- [BME280 \(i2c\\_inst\\_t \\*i2cPort, uint8\\_t address=ADDR\\_SDO\\_LOW\)](#)
- [bool init \(\)](#)
- [void reset \(\)](#)
- [bool read\\_raw\\_all \(int32\\_t \\*temperature, int32\\_t \\*pressure, int32\\_t \\*humidity\)](#)
- [float convert\\_temperature \(int32\\_t temp\\_raw\) const](#)
- [float convert\\_pressure \(int32\\_t pressure\\_raw\) const](#)
- [float convert\\_humidity \(int32\\_t humidity\\_raw\) const](#)

### Static Public Attributes

- [static constexpr uint8\\_t ADDR\\_SDO\\_LOW = 0x76](#)
- [static constexpr uint8\\_t ADDR\\_SDO\\_HIGH = 0x77](#)

### Private Member Functions

- [bool configure\\_sensor \(\)](#)
- [bool get\\_calibration\\_parameters \(\)](#)

### Private Attributes

- `i2c_inst_t * i2c_port`
- `uint8_t device_addr`
- `BME280CalibParam calib_params`
- `bool initialized_`
- `int32_t t_fine`

### Static Private Attributes

- `static constexpr uint8_t REG_CONFIG = 0xF5`
- `static constexpr uint8_t REG_CTRL_MEAS = 0xF4`
- `static constexpr uint8_t REG_CTRL_HUM = 0xF2`
- `static constexpr uint8_t REG_RESET = 0xE0`
- `static constexpr uint8_t REG_PRESSURE_MSB = 0xF7`
- `static constexpr uint8_t REG_TEMPERATURE_MSB = 0xFA`
- `static constexpr uint8_t REG_HUMIDITY_MSB = 0xFD`
- `static constexpr uint8_t REG_DIG_T1_LSB = 0x88`
- `static constexpr uint8_t REG_DIG_T1_MSB = 0x89`
- `static constexpr uint8_t REG_DIG_T2_LSB = 0x8A`
- `static constexpr uint8_t REG_DIG_T2_MSB = 0x8B`
- `static constexpr uint8_t REG_DIG_T3_LSB = 0x8C`
- `static constexpr uint8_t REG_DIG_T3_MSB = 0x8D`
- `static constexpr uint8_t REG_DIG_P1_LSB = 0x8E`
- `static constexpr uint8_t REG_DIG_P1_MSB = 0x8F`
- `static constexpr uint8_t REG_DIG_P2_LSB = 0x90`
- `static constexpr uint8_t REG_DIG_P2_MSB = 0x91`
- `static constexpr uint8_t REG_DIG_P3_LSB = 0x92`
- `static constexpr uint8_t REG_DIG_P3_MSB = 0x93`
- `static constexpr uint8_t REG_DIG_P4_LSB = 0x94`
- `static constexpr uint8_t REG_DIG_P4_MSB = 0x95`
- `static constexpr uint8_t REG_DIG_P5_LSB = 0x96`
- `static constexpr uint8_t REG_DIG_P5_MSB = 0x97`
- `static constexpr uint8_t REG_DIG_P6_LSB = 0x98`
- `static constexpr uint8_t REG_DIG_P6_MSB = 0x99`
- `static constexpr uint8_t REG_DIG_P7_LSB = 0x9A`
- `static constexpr uint8_t REG_DIG_P7_MSB = 0x9B`
- `static constexpr uint8_t REG_DIG_P8_LSB = 0x9C`
- `static constexpr uint8_t REG_DIG_P8_MSB = 0x9D`
- `static constexpr uint8_t REG_DIG_P9_LSB = 0x9E`
- `static constexpr uint8_t REG_DIG_P9_MSB = 0x9F`
- `static constexpr uint8_t REG_DIG_H1 = 0xA1`
- `static constexpr uint8_t REG_DIG_H2 = 0xE1`
- `static constexpr uint8_t REG_DIG_H3 = 0xE3`
- `static constexpr uint8_t REG_DIG_H4 = 0xE4`
- `static constexpr uint8_t REG_DIG_H5 = 0xE5`
- `static constexpr uint8_t REG_DIG_H6 = 0xE7`
- `static constexpr size_t NUM_CALIB_PARAMS = 24`

### 7.3.1 Detailed Description

Definition at line 38 of file [BME280.h](#).

### 7.3.2 Constructor & Destructor Documentation

#### 7.3.2.1 BME280()

```
BME280::BME280 (
    i2c_inst_t * i2cPort,
    uint8_t address = ADDR_SDO_LOW)
```

Definition at line 14 of file [BME280.cpp](#).

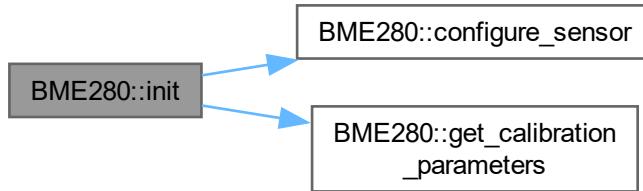
### 7.3.3 Member Function Documentation

#### 7.3.3.1 init()

```
bool BME280::init ()
```

Definition at line 18 of file [BME280.cpp](#).

Here is the call graph for this function:



#### 7.3.3.2 reset()

```
void BME280::reset ()
```

Definition at line 59 of file [BME280.cpp](#).

#### 7.3.3.3 read\_raw\_all()

```
bool BME280::read_raw_all (
    int32_t * temperature,
    int32_t * pressure,
    int32_t * humidity)
```

Definition at line 68 of file [BME280.cpp](#).

#### 7.3.3.4 convert\_temperature()

```
float BME280::convert_temperature (
    int32_t temp_raw) const
```

Definition at line 101 of file [BME280.cpp](#).

#### 7.3.3.5 convert\_pressure()

```
float BME280::convert_pressure (
    int32_t pressure_raw) const
```

Definition at line 110 of file [BME280.cpp](#).

#### 7.3.3.6 convert\_humidity()

```
float BME280::convert_humidity (
    int32_t humidity_raw) const
```

Definition at line 131 of file [BME280.cpp](#).

#### 7.3.3.7 configure\_sensor()

```
bool BME280::configure_sensor () [private]
```

Definition at line 201 of file [BME280.cpp](#).

Here is the caller graph for this function:



#### 7.3.3.8 get\_calibration\_parameters()

```
bool BME280::get_calibration_parameters () [private]
```

Definition at line 143 of file [BME280.cpp](#).

Here is the caller graph for this function:



### 7.3.4 Member Data Documentation

#### 7.3.4.1 ADDR\_SDO\_LOW

```
uint8_t BME280::ADDR_SDO_LOW = 0x76 [static], [constexpr]
```

Definition at line 41 of file [BME280.h](#).

#### 7.3.4.2 ADDR\_SDO\_HIGH

```
uint8_t BME280::ADDR_SDO_HIGH = 0x77 [static], [constexpr]
```

Definition at line 42 of file [BME280.h](#).

#### 7.3.4.3 i2c\_port

```
i2c_inst_t* BME280::i2c_port [private]
```

Definition at line 69 of file [BME280.h](#).

#### 7.3.4.4 device\_addr

```
uint8_t BME280::device_addr [private]
```

Definition at line 70 of file [BME280.h](#).

#### 7.3.4.5 calib\_params

```
BME280CalibParam BME280::calib_params [private]
```

Definition at line 73 of file [BME280.h](#).

#### 7.3.4.6 initialized\_

```
bool BME280::initialized_ [private]
```

Definition at line 76 of file [BME280.h](#).

#### 7.3.4.7 t\_fine

```
int32_t BME280::t_fine [mutable], [private]
```

Definition at line 79 of file [BME280.h](#).

### 7.3.4.8 REG\_CONFIG

```
uint8_t BME280::REG_CONFIG = 0xF5 [static], [constexpr], [private]
```

Definition at line [82](#) of file [BME280.h](#).

### 7.3.4.9 REG\_CTRL\_MEAS

```
uint8_t BME280::REG_CTRL_MEAS = 0xF4 [static], [constexpr], [private]
```

Definition at line [83](#) of file [BME280.h](#).

### 7.3.4.10 REG\_CTRL\_HUM

```
uint8_t BME280::REG_CTRL_HUM = 0xF2 [static], [constexpr], [private]
```

Definition at line [84](#) of file [BME280.h](#).

### 7.3.4.11 REG\_RESET

```
uint8_t BME280::REG_RESET = 0xE0 [static], [constexpr], [private]
```

Definition at line [85](#) of file [BME280.h](#).

### 7.3.4.12 REG\_PRESSURE\_MSB

```
uint8_t BME280::REG_PRESSURE_MSB = 0xF7 [static], [constexpr], [private]
```

Definition at line [87](#) of file [BME280.h](#).

### 7.3.4.13 REG\_TEMPERATURE\_MSB

```
uint8_t BME280::REG_TEMPERATURE_MSB = 0xFA [static], [constexpr], [private]
```

Definition at line [88](#) of file [BME280.h](#).

### 7.3.4.14 REG\_HUMIDITY\_MSB

```
uint8_t BME280::REG_HUMIDITY_MSB = 0xFD [static], [constexpr], [private]
```

Definition at line [89](#) of file [BME280.h](#).

### 7.3.4.15 REG\_DIG\_T1\_LSB

```
uint8_t BME280::REG_DIG_T1_LSB = 0x88 [static], [constexpr], [private]
```

Definition at line [92](#) of file [BME280.h](#).

#### 7.3.4.16 REG\_DIG\_T1\_MSB

```
uint8_t BME280::REG_DIG_T1_MSB = 0x89 [static], [constexpr], [private]
```

Definition at line 93 of file [BME280.h](#).

#### 7.3.4.17 REG\_DIG\_T2\_LSB

```
uint8_t BME280::REG_DIG_T2_LSB = 0x8A [static], [constexpr], [private]
```

Definition at line 94 of file [BME280.h](#).

#### 7.3.4.18 REG\_DIG\_T2\_MSB

```
uint8_t BME280::REG_DIG_T2_MSB = 0x8B [static], [constexpr], [private]
```

Definition at line 95 of file [BME280.h](#).

#### 7.3.4.19 REG\_DIG\_T3\_LSB

```
uint8_t BME280::REG_DIG_T3_LSB = 0x8C [static], [constexpr], [private]
```

Definition at line 96 of file [BME280.h](#).

#### 7.3.4.20 REG\_DIG\_T3\_MSB

```
uint8_t BME280::REG_DIG_T3_MSB = 0x8D [static], [constexpr], [private]
```

Definition at line 97 of file [BME280.h](#).

#### 7.3.4.21 REG\_DIG\_P1\_LSB

```
uint8_t BME280::REG_DIG_P1_LSB = 0x8E [static], [constexpr], [private]
```

Definition at line 99 of file [BME280.h](#).

#### 7.3.4.22 REG\_DIG\_P1\_MSB

```
uint8_t BME280::REG_DIG_P1_MSB = 0x8F [static], [constexpr], [private]
```

Definition at line 100 of file [BME280.h](#).

#### 7.3.4.23 REG\_DIG\_P2\_LSB

```
uint8_t BME280::REG_DIG_P2_LSB = 0x90 [static], [constexpr], [private]
```

Definition at line 101 of file [BME280.h](#).

#### 7.3.4.24 REG\_DIG\_P2\_MSB

```
uint8_t BME280::REG_DIG_P2_MSB = 0x91 [static], [constexpr], [private]
```

Definition at line 102 of file [BME280.h](#).

#### 7.3.4.25 REG\_DIG\_P3\_LSB

```
uint8_t BME280::REG_DIG_P3_LSB = 0x92 [static], [constexpr], [private]
```

Definition at line 103 of file [BME280.h](#).

#### 7.3.4.26 REG\_DIG\_P3\_MSB

```
uint8_t BME280::REG_DIG_P3_MSB = 0x93 [static], [constexpr], [private]
```

Definition at line 104 of file [BME280.h](#).

#### 7.3.4.27 REG\_DIG\_P4\_LSB

```
uint8_t BME280::REG_DIG_P4_LSB = 0x94 [static], [constexpr], [private]
```

Definition at line 105 of file [BME280.h](#).

#### 7.3.4.28 REG\_DIG\_P4\_MSB

```
uint8_t BME280::REG_DIG_P4_MSB = 0x95 [static], [constexpr], [private]
```

Definition at line 106 of file [BME280.h](#).

#### 7.3.4.29 REG\_DIG\_P5\_LSB

```
uint8_t BME280::REG_DIG_P5_LSB = 0x96 [static], [constexpr], [private]
```

Definition at line 107 of file [BME280.h](#).

#### 7.3.4.30 REG\_DIG\_P5\_MSB

```
uint8_t BME280::REG_DIG_P5_MSB = 0x97 [static], [constexpr], [private]
```

Definition at line 108 of file [BME280.h](#).

#### 7.3.4.31 REG\_DIG\_P6\_LSB

```
uint8_t BME280::REG_DIG_P6_LSB = 0x98 [static], [constexpr], [private]
```

Definition at line 109 of file [BME280.h](#).

### 7.3.4.32 REG\_DIG\_P6\_MSB

```
uint8_t BME280::REG_DIG_P6_MSB = 0x99 [static], [constexpr], [private]
```

Definition at line 110 of file [BME280.h](#).

### 7.3.4.33 REG\_DIG\_P7\_LSB

```
uint8_t BME280::REG_DIG_P7_LSB = 0x9A [static], [constexpr], [private]
```

Definition at line 111 of file [BME280.h](#).

### 7.3.4.34 REG\_DIG\_P7\_MSB

```
uint8_t BME280::REG_DIG_P7_MSB = 0x9B [static], [constexpr], [private]
```

Definition at line 112 of file [BME280.h](#).

### 7.3.4.35 REG\_DIG\_P8\_LSB

```
uint8_t BME280::REG_DIG_P8_LSB = 0x9C [static], [constexpr], [private]
```

Definition at line 113 of file [BME280.h](#).

### 7.3.4.36 REG\_DIG\_P8\_MSB

```
uint8_t BME280::REG_DIG_P8_MSB = 0x9D [static], [constexpr], [private]
```

Definition at line 114 of file [BME280.h](#).

### 7.3.4.37 REG\_DIG\_P9\_LSB

```
uint8_t BME280::REG_DIG_P9_LSB = 0x9E [static], [constexpr], [private]
```

Definition at line 115 of file [BME280.h](#).

### 7.3.4.38 REG\_DIG\_P9\_MSB

```
uint8_t BME280::REG_DIG_P9_MSB = 0x9F [static], [constexpr], [private]
```

Definition at line 116 of file [BME280.h](#).

### 7.3.4.39 REG\_DIG\_H1

```
uint8_t BME280::REG_DIG_H1 = 0xA1 [static], [constexpr], [private]
```

Definition at line 119 of file [BME280.h](#).

### 7.3.4.40 REG\_DIG\_H2

```
uint8_t BME280::REG_DIG_H2 = 0xE1 [static], [constexpr], [private]
```

Definition at line 120 of file [BME280.h](#).

### 7.3.4.41 REG\_DIG\_H3

```
uint8_t BME280::REG_DIG_H3 = 0xE3 [static], [constexpr], [private]
```

Definition at line 121 of file [BME280.h](#).

### 7.3.4.42 REG\_DIG\_H4

```
uint8_t BME280::REG_DIG_H4 = 0xE4 [static], [constexpr], [private]
```

Definition at line 122 of file [BME280.h](#).

### 7.3.4.43 REG\_DIG\_H5

```
uint8_t BME280::REG_DIG_H5 = 0xE5 [static], [constexpr], [private]
```

Definition at line 123 of file [BME280.h](#).

### 7.3.4.44 REG\_DIG\_H6

```
uint8_t BME280::REG_DIG_H6 = 0xE7 [static], [constexpr], [private]
```

Definition at line 124 of file [BME280.h](#).

### 7.3.4.45 NUM\_CALIB\_PARAMS

```
size_t BME280::NUM_CALIB_PARAMS = 24 [static], [constexpr], [private]
```

Definition at line 127 of file [BME280.h](#).

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

- lib/sensors/BME280/[BME280.h](#)
- lib/sensors/BME280/[BME280.cpp](#)

## 7.4 BME280CalibParam Struct Reference

```
#include <BME280.h>
```

## Public Attributes

- `uint16_t dig_t1`
- `int16_t dig_t2`
- `int16_t dig_t3`
- `uint16_t dig_p1`
- `int16_t dig_p2`
- `int16_t dig_p3`
- `int16_t dig_p4`
- `int16_t dig_p5`
- `int16_t dig_p6`
- `int16_t dig_p7`
- `int16_t dig_p8`
- `int16_t dig_p9`
- `uint8_t dig_h1`
- `int16_t dig_h2`
- `uint8_t dig_h3`
- `int16_t dig_h4`
- `int16_t dig_h5`
- `int8_t dig_h6`

### 7.4.1 Detailed Description

Definition at line 11 of file [BME280.h](#).

### 7.4.2 Member Data Documentation

#### 7.4.2.1 `dig_t1`

```
uint16_t BME280CalibParam::dig_t1
```

Definition at line 13 of file [BME280.h](#).

#### 7.4.2.2 `dig_t2`

```
int16_t BME280CalibParam::dig_t2
```

Definition at line 14 of file [BME280.h](#).

#### 7.4.2.3 `dig_t3`

```
int16_t BME280CalibParam::dig_t3
```

Definition at line 15 of file [BME280.h](#).

#### 7.4.2.4 `dig_p1`

```
uint16_t BME280CalibParam::dig_p1
```

Definition at line 18 of file [BME280.h](#).

#### 7.4.2.5 `dig_p2`

```
int16_t BME280CalibParam::dig_p2
```

Definition at line 19 of file [BME280.h](#).

#### 7.4.2.6 `dig_p3`

```
int16_t BME280CalibParam::dig_p3
```

Definition at line 20 of file [BME280.h](#).

#### 7.4.2.7 `dig_p4`

```
int16_t BME280CalibParam::dig_p4
```

Definition at line 21 of file [BME280.h](#).

#### 7.4.2.8 `dig_p5`

```
int16_t BME280CalibParam::dig_p5
```

Definition at line 22 of file [BME280.h](#).

#### 7.4.2.9 `dig_p6`

```
int16_t BME280CalibParam::dig_p6
```

Definition at line 23 of file [BME280.h](#).

#### 7.4.2.10 `dig_p7`

```
int16_t BME280CalibParam::dig_p7
```

Definition at line 24 of file [BME280.h](#).

#### 7.4.2.11 `dig_p8`

```
int16_t BME280CalibParam::dig_p8
```

Definition at line 25 of file [BME280.h](#).

#### 7.4.2.12 dig\_p9

```
int16_t BME280CalibParam::dig_p9
```

Definition at line 26 of file [BME280.h](#).

#### 7.4.2.13 dig\_h1

```
uint8_t BME280CalibParam::dig_h1
```

Definition at line 29 of file [BME280.h](#).

#### 7.4.2.14 dig\_h2

```
int16_t BME280CalibParam::dig_h2
```

Definition at line 30 of file [BME280.h](#).

#### 7.4.2.15 dig\_h3

```
uint8_t BME280CalibParam::dig_h3
```

Definition at line 31 of file [BME280.h](#).

#### 7.4.2.16 dig\_h4

```
int16_t BME280CalibParam::dig_h4
```

Definition at line 32 of file [BME280.h](#).

#### 7.4.2.17 dig\_h5

```
int16_t BME280CalibParam::dig_h5
```

Definition at line 33 of file [BME280.h](#).

#### 7.4.2.18 dig\_h6

```
int8_t BME280CalibParam::dig_h6
```

Definition at line 34 of file [BME280.h](#).

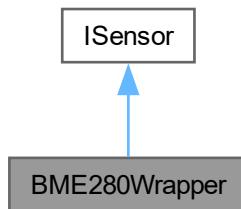
The documentation for this struct was generated from the following file:

- lib/sensors/BME280/[BME280.h](#)

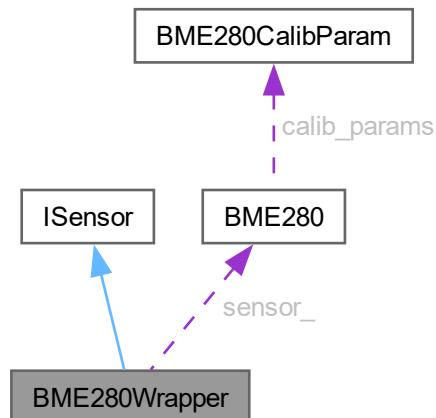
## 7.5 BME280Wrapper Class Reference

```
#include <BME280_WRAPPER.h>
```

Inheritance diagram for BME280Wrapper:



Collaboration diagram for BME280Wrapper:



### Public Member Functions

- `BME280Wrapper (i2c_inst_t *i2c)`
- `bool init () override`
- `float read_data (SensorDataTypelIdentifier type) override`
- `bool is_initialized () const override`
- `SensorType get_type () const override`
- `bool configure (const std::map< std::string, std::string > &config) override`
- `uint8_t get_address () const override`

## Public Member Functions inherited from [ISensor](#)

- virtual [~ISensor](#) ()=default

## Private Attributes

- [BME280 sensor\\_](#)
- bool [initialized\\_](#) = false

### 7.5.1 Detailed Description

Definition at line 8 of file [BME280\\_WRAPPER.h](#).

### 7.5.2 Constructor & Destructor Documentation

#### 7.5.2.1 [BME280Wrapper\(\)](#)

```
BME280Wrapper::BME280Wrapper (
    i2c_inst_t * i2c)
```

Definition at line 3 of file [BME280\\_WRAPPER.cpp](#).

### 7.5.3 Member Function Documentation

#### 7.5.3.1 [init\(\)](#)

```
bool BME280Wrapper::init () [override], [virtual]
```

Implements [ISensor](#).

Definition at line 5 of file [BME280\\_WRAPPER.cpp](#).

#### 7.5.3.2 [read\\_data\(\)](#)

```
float BME280Wrapper::read_data (
    SensorDataTypeIdentifier type) [override], [virtual]
```

Implements [ISensor](#).

Definition at line 10 of file [BME280\\_WRAPPER.cpp](#).

#### 7.5.3.3 [is\\_initialized\(\)](#)

```
bool BME280Wrapper::is_initialized () const [override], [virtual]
```

Implements [ISensor](#).

Definition at line 26 of file [BME280\\_WRAPPER.cpp](#).

#### 7.5.3.4 `get_type()`

```
SensorType BME280Wrapper::get_type () const [override], [virtual]
```

Implements [ISensor](#).

Definition at line 30 of file [BME280\\_WRAPPER.cpp](#).

#### 7.5.3.5 `configure()`

```
bool BME280Wrapper::configure (
    const std::map< std::string, std::string > & config) [override], [virtual]
```

Implements [ISensor](#).

Definition at line 34 of file [BME280\\_WRAPPER.cpp](#).

#### 7.5.3.6 `get_address()`

```
uint8_t BME280Wrapper::get_address () const [inline], [override], [virtual]
```

Implements [ISensor](#).

Definition at line 22 of file [BME280\\_WRAPPER.h](#).

### 7.5.4 Member Data Documentation

#### 7.5.4.1 `sensor_`

```
BME280 BME280Wrapper::sensor_ [private]
```

Definition at line 10 of file [BME280\\_WRAPPER.h](#).

#### 7.5.4.2 `initialized_`

```
bool BME280Wrapper::initialized_ = false [private]
```

Definition at line 11 of file [BME280\\_WRAPPER.h](#).

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

- lib/sensors/BME280/[BME280\\_WRAPPER.h](#)
- lib/sensors/BME280/[BME280\\_WRAPPER.cpp](#)

## 7.6 INA3221::conf\_reg\_t Struct Reference

Configuration register bit fields.

## Public Attributes

- `uint16_t mode_shunt_en:1`
- `uint16_t mode_bus_en:1`
- `uint16_t mode_continious_en:1`
- `uint16_t shunt_conv_time:3`
- `uint16_t bus_conv_time:3`
- `uint16_t avg_mode:3`
- `uint16_t ch3_en:1`
- `uint16_t ch2_en:1`
- `uint16_t ch1_en:1`
- `uint16_t reset:1`

### 7.6.1 Detailed Description

Configuration register bit fields.

Definition at line 101 of file [INA3221.h](#).

### 7.6.2 Member Data Documentation

#### 7.6.2.1 mode\_shunt\_en

```
uint16_t INA3221::conf_reg_t::mode_shunt_en
```

Definition at line 102 of file [INA3221.h](#).

#### 7.6.2.2 mode\_bus\_en

```
uint16_t INA3221::conf_reg_t::mode_bus_en
```

Definition at line 103 of file [INA3221.h](#).

#### 7.6.2.3 mode\_continious\_en

```
uint16_t INA3221::conf_reg_t::mode_continious_en
```

Definition at line 104 of file [INA3221.h](#).

#### 7.6.2.4 shunt\_conv\_time

```
uint16_t INA3221::conf_reg_t::shunt_conv_time
```

Definition at line 105 of file [INA3221.h](#).

### 7.6.2.5 bus\_conv\_time

```
uint16_t INA3221::conf_reg_t::bus_conv_time
```

Definition at line 106 of file [INA3221.h](#).

### 7.6.2.6 avg\_mode

```
uint16_t INA3221::conf_reg_t::avg_mode
```

Definition at line 107 of file [INA3221.h](#).

### 7.6.2.7 ch3\_en

```
uint16_t INA3221::conf_reg_t::ch3_en
```

Definition at line 108 of file [INA3221.h](#).

### 7.6.2.8 ch2\_en

```
uint16_t INA3221::conf_reg_t::ch2_en
```

Definition at line 109 of file [INA3221.h](#).

### 7.6.2.9 ch1\_en

```
uint16_t INA3221::conf_reg_t::ch1_en
```

Definition at line 110 of file [INA3221.h](#).

### 7.6.2.10 reset

```
uint16_t INA3221::conf_reg_t::reset
```

Definition at line 111 of file [INA3221.h](#).

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

- lib/powerman/INA3221/[INA3221.h](#)

## 7.7 DS3231 Class Reference

Class for interfacing with the [DS3231](#) real-time clock.

```
#include <DS3231.h>
```

## Public Member Functions

- `DS3231 (i2c_inst_t *i2c_instance)`  
*Constructor for the `DS3231` class.*
- `int set_time (ds3231_data_t *data)`  
*Sets the time on the `DS3231` clock.*
- `int get_time (ds3231_data_t *data)`  
*Gets the current time from the `DS3231` clock.*
- `int read_temperature (float *resolution)`  
*Reads the current temperature from the `DS3231`.*
- `int set_unix_time (time_t unix_time)`  
*Sets the time using a Unix timestamp.*
- `time_t get_unix_time ()`  
*Gets the current time as a Unix timestamp.*
- `int clock_enable ()`  
*Enables the `DS3231` clock oscillator.*
- `int16_t get_timezone_offset () const`  
*Gets the current timezone offset.*
- `void set_timezone_offset (int16_t offset_minutes)`  
*Sets the timezone offset.*
- `uint32_t get_clock_sync_interval () const`  
*Gets the clock synchronization interval.*
- `void set_clock_sync_interval (uint32_t interval_minutes)`  
*Sets the clock synchronization interval.*
- `time_t get_last_sync_time () const`  
*Gets the timestamp of the last clock synchronization.*
- `void update_last_sync_time ()`  
*Updates the last sync time to current time.*
- `time_t get_local_time ()`  
*Gets the current local time (including timezone offset)*
- `bool is_sync_needed ()`  
*Checks if clock synchronization is needed.*
- `bool sync_clock_with_gps ()`  
*Synchronizes clock with GPS data.*

## Private Member Functions

- `int i2c_read_reg (uint8_t reg_addr, size_t length, uint8_t *data)`  
*Reads data from a specific register on the `DS3231`.*
- `int i2c_write_reg (uint8_t reg_addr, size_t length, uint8_t *data)`  
*Writes data to a specific register on the `DS3231`.*
- `uint8_t bin_to_bcd (const uint8_t data)`  
*Converts binary value to BCD (Binary Coded Decimal)*
- `uint8_t bcd_to_bin (const uint8_t bcd)`  
*Converts BCD (Binary Coded Decimal) to binary value.*

## Private Attributes

- `i2c_inst_t * i2c`  
`Pointer to I2C instance.`
- `uint8_t ds3231_addr`  
`I2C address of DS3231.`
- `recursive_mutex_t clock_mutex_`  
`Mutex for thread-safe I2C access.`
- `int16_t timezone_offset_minutes_ = 60`  
`Timezone offset in minutes, default: UTC.`
- `uint32_t sync_interval_minutes_ = 1440`  
`Sync interval in minutes, default: 24 hours.`
- `time_t last_sync_time_ = 0`  
`Last sync timestamp, 0 = never synced.`

## 7.7.1 Detailed Description

Class for interfacing with the [DS3231](#) real-time clock.

This class provides methods to set and get time from a [DS3231](#) RTC module, handle timezone offsets, perform clock synchronization, and more.

Definition at line 108 of file [DS3231.h](#).

## 7.7.2 Constructor & Destructor Documentation

### 7.7.2.1 DS3231()

```
DS3231::DS3231 (
    i2c_inst_t * i2c_instance)
```

Constructor for the [DS3231](#) class.

#### Parameters

in	<code>i2c_instance</code>	Pointer to the I2C instance to use
in	<code>i2c_instance</code>	Pointer to the <code>i2c_inst_t</code> structure representing the I <sup>2</sup> C port

Initializes a [DS3231](#) RTC controller object with the specified I<sup>2</sup>C port and default device address. This constructor also initializes the clock mutex for thread-safe I<sup>2</sup>C access.

Definition at line 16 of file [DS3231.cpp](#).

## 7.7.3 Member Function Documentation

### 7.7.3.1 set\_time()

```
int DS3231::set_time (
    ds3231_data_t * data)
```

Sets the time on the [DS3231](#) clock.

Sets the time on the [DS3231](#) RTC module.

**Parameters**

in	<i>data</i>	Pointer to a <a href="#">ds3231_data_t</a> structure with time information
----	-------------	--

**Returns**

0 on success, -1 on failure

**Parameters**

in	<i>data</i>	Pointer to a <a href="#">ds3231_data_t</a> structure containing the time to set
----	-------------	---

**Returns**

0 on success, -1 on failure

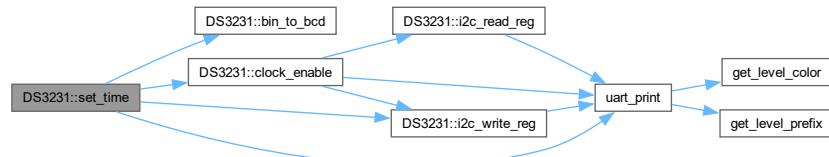
Writes time and date data to the [DS3231](#) module. The function performs input validation to ensure that the provided values are within valid ranges. The time values are converted from binary to BCD format before being written to the device registers.

**Note**

The [ds3231\\_data\\_t](#) structure must contain valid values for seconds, minutes, hours, day, date, month, year, and century.

Definition at line 33 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**7.7.3.2 `get_time()`**

```
int DS3231::get_time (
    ds3231_data_t * data)
```

Gets the current time from the [DS3231](#) clock.

Reads the current time from the [DS3231](#) RTC module.

**Parameters**

<code>out</code>	<code>data</code>	Pointer to a <a href="#">ds3231_data_t</a> structure to store time information
------------------	-------------------	--

**Returns**

0 on success, -1 on failure

**Parameters**

<code>out</code>	<code>data</code>	Pointer to a <a href="#">ds3231_data_t</a> structure to store the read time
------------------	-------------------	---

**Returns**

0 on success, -1 on failure

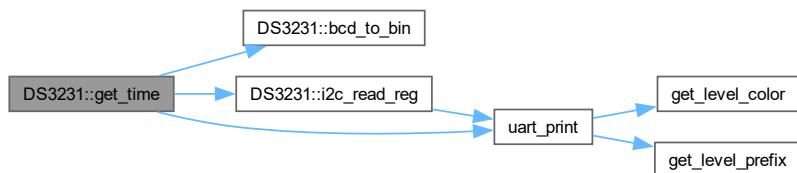
Reads the time and date registers from the [DS3231](#) and stores the decoded values in the provided data structure. The BCD values from the registers are converted to binary format. The function performs validation on the read values to ensure they are within valid ranges.

**Note**

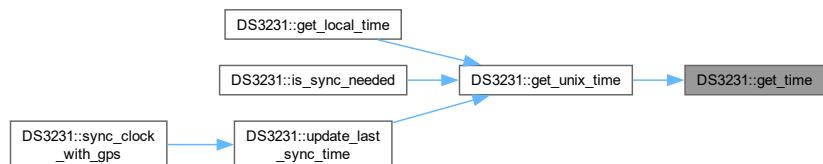
The function logs debug information including the raw BCD values read and the decoded time and date.

Definition at line 102 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.7.3.3 `read_temperature()`

```
int DS3231::read_temperature (
    float * resolution)
```

Reads the current temperature from the [DS3231](#).

Reads the temperature from the [DS3231](#)'s internal temperature sensor.

**Parameters**

<code>out</code>	<code>resolution</code>	Pointer to store the temperature value in Celsius
------------------	-------------------------	---

**Returns**

0 on success, -1 on failure

**Parameters**

<code>out</code>	<code>resolution</code>	Pointer to a float to store the temperature value
------------------	-------------------------	---

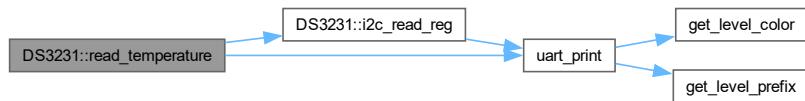
**Returns**

0 on success, -1 on failure

The [DS3231](#) includes an internal temperature sensor with 0.25°C resolution. This function reads the sensor value and calculates the temperature in degrees Celsius. The temperature sensor is primarily used for the oscillator's temperature compensation, but can be used for general temperature monitoring as well.

Definition at line 155 of file [DS3231.cpp](#).

Here is the call graph for this function:

**7.7.3.4 set\_unix\_time()**

```
int DS3231::set_unix_time (
    time_t unix_time)
```

Sets the time using a Unix timestamp.

Sets the [DS3231](#) clock using a Unix timestamp.

**Parameters**

<code>in</code>	<code>unix_time</code>	Time as seconds since Unix epoch (1970-01-01 00:00:00 UTC)
-----------------	------------------------	--

**Returns**

0 on success, -1 on failure

**Parameters**

in	<i>unix_time</i>	The time in seconds since the Unix epoch (1970-01-01 00:00:00 UTC)
----	------------------	--

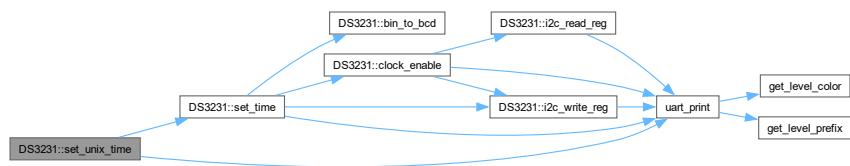
**Returns**

0 on success, -1 on failure

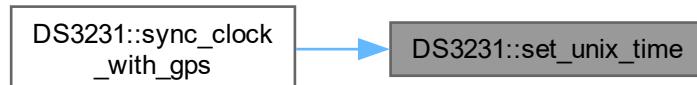
Converts the provided Unix timestamp to a calendar date and time and sets the [DS3231](#) RTC accordingly. This function properly handles the conversion between the tm structure (used by C standard library) and the internal [ds3231\\_data\\_t](#) format.

Definition at line 184 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**7.7.3.5 `get_unix_time()`**

```
time_t DS3231::get_unix_time ()
```

Gets the current time as a Unix timestamp.

Gets the current time from [DS3231](#) as a Unix timestamp.

**Returns**

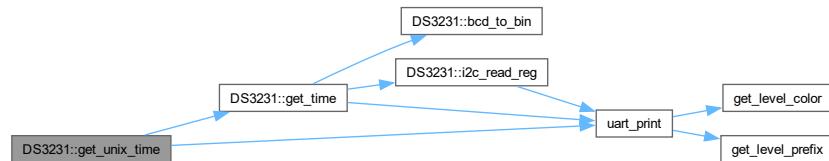
Unix timestamp, or -1 on error

Unix timestamp (seconds since 1970-01-01 00:00:00 UTC), or -1 on error

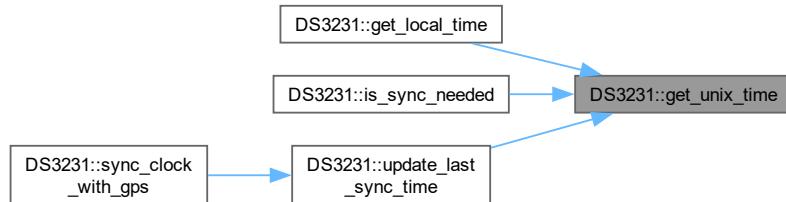
Reads the current time from the [DS3231](#) RTC and converts it to a Unix timestamp. This function properly handles the conversion between the internal [ds3231\\_data\\_t](#) format and the tm structure used by the C standard library.

Definition at line 214 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**7.7.3.6 `clock_enable()`**

```
int DS3231::clock_enable ()
```

Enables the [DS3231](#) clock oscillator.

Enables the [DS3231](#)'s oscillator.

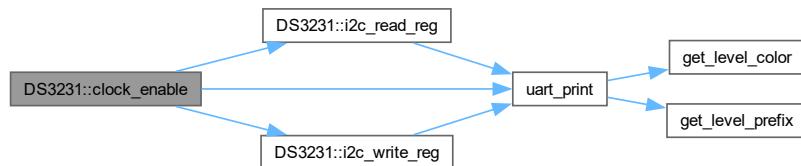
**Returns**

0 on success, -1 on failure  
 0 on success, -1 on failure

Reads the control register and clears the EOSC (Enable Oscillator) bit to ensure the oscillator is running. This is necessary for the RTC to keep time when not on external power.

Definition at line 251 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**7.7.3.7 get\_timezone\_offset()**

```
int16_t DS3231::get_timezone_offset () const
```

Gets the current timezone offset.

Gets the currently configured timezone offset.

**Returns**

Timezone offset in minutes (-720 to +720)  
 The timezone offset in minutes

Returns the current timezone offset in minutes relative to UTC. Positive values represent timezones ahead of UTC (east), negative values represent timezones behind UTC (west).

Definition at line 283 of file [DS3231.cpp](#).

**7.7.3.8 set\_timezone\_offset()**

```
void DS3231::set_timezone_offset (
    int16_t offset_minutes)
```

Sets the timezone offset.

**Parameters**

	<i>offset_minutes</i>	Offset in minutes (-720 to +720)
in	<i>offset_minutes</i>	The timezone offset in minutes

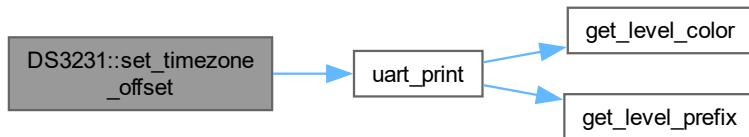
Sets the timezone offset in minutes relative to UTC. This value is used when converting between UTC and local time. The function validates that the offset is within a valid range (-720 to +720 minutes, which corresponds to -12 to +12 hours).

**Note**

This setting is stored in memory and does not persist across reboots.

Definition at line 298 of file [DS3231.cpp](#).

Here is the call graph for this function:

**7.7.3.9 get\_clock\_sync\_interval()**

```
uint32_t DS3231::get_clock_sync_interval () const
```

Gets the clock synchronization interval.

Gets the currently configured clock synchronization interval.

**Returns**

Sync interval in minutes

The sync interval in minutes

Returns the current interval between clock synchronization attempts. This is the time after which [is\\_sync\\_needed\(\)](#) will return true.

Definition at line 315 of file [DS3231.cpp](#).

**7.7.3.10 set\_clock\_sync\_interval()**

```
void DS3231::set_clock_sync_interval (
    uint32_t interval_minutes)
```

Sets the clock synchronization interval.

**Parameters**

	<i>interval_minutes</i>	Interval in minutes (1-43200)
in	<i>interval_minutes</i>	The desired sync interval in minutes

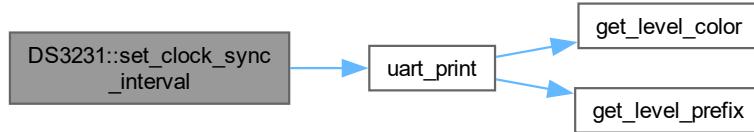
Sets how frequently the clock should be synchronized with an external time source (such as GPS). The function validates that the interval is within a valid range (1 minute to 43200 minutes, which is 30 days).

**Note**

This setting is stored in memory and does not persist across reboots.

Definition at line 329 of file [DS3231.cpp](#).

Here is the call graph for this function:

**7.7.3.11 get\_last\_sync\_time()**

```
time_t DS3231::get_last_sync_time () const
```

Gets the timestamp of the last clock synchronization.

Gets the timestamp of the last successful clock synchronization.

**Returns**

Unix timestamp of last sync, 0 if never synced

Unix timestamp of the last sync, or 0 if never synced

Returns the Unix timestamp of when the clock was last successfully synchronized with an external time source. A value of 0 indicates that the clock has never been synchronized.

Definition at line 346 of file [DS3231.cpp](#).

### 7.7.3.12 update\_last\_sync\_time()

```
void DS3231::update_last_sync_time ()
```

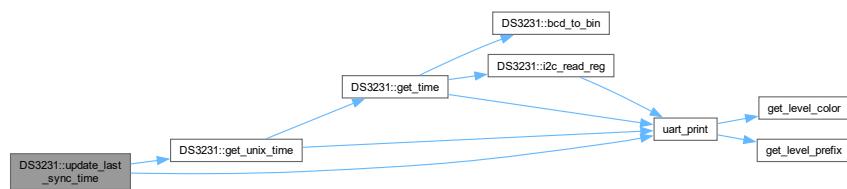
Updates the last sync time to current time.

Updates the last sync timestamp to the current time.

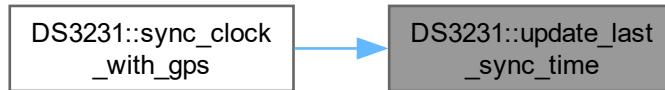
Records the current time as the last successful synchronization time. This should be called after successfully setting the time from an external source (such as GPS). The function logs the update with an informational message.

Definition at line 359 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.7.3.13 get\_local\_time()

```
time_t DS3231::get_local_time ()
```

Gets the current local time (including timezone offset)

Gets the current local time by applying the timezone offset to UTC time.

**Returns**

Unix timestamp adjusted for timezone, or -1 on error  
 Local time as Unix timestamp, or -1 on error

Retrieves the current UTC time from the RTC and applies the configured timezone offset (in minutes) to calculate the local time.

Definition at line 372 of file [DS3231.cpp](#).

Here is the call graph for this function:

**7.7.3.14 is\_sync\_needed()**

```
bool DS3231::is_sync_needed ()
```

Checks if clock synchronization is needed.

Determines if the clock needs synchronization based on the configured interval.

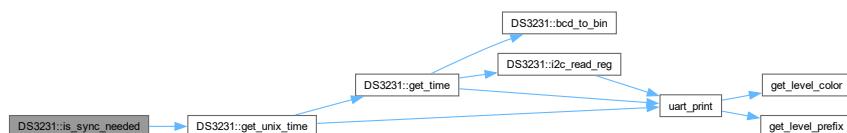
**Returns**

true if sync interval has elapsed since last sync, false otherwise  
 true if synchronization is needed, false otherwise

This method checks if the clock has ever been synchronized (`last_sync_time_` is 0) or if the time elapsed since the last synchronization exceeds the configured `sync_interval_minutes_`. If the current time cannot be determined, it assumes synchronization is needed.

Definition at line 391 of file [DS3231.cpp](#).

Here is the call graph for this function:

**7.7.3.15 sync\_clock\_with\_gps()**

```
bool DS3231::sync_clock_with_gps ()
```

Synchronizes clock with GPS data.

Synchronizes the RTC with time from GPS.

**Returns**

true if sync successful, false otherwise

## Parameters

in *nmea\_data* Reference to NMEA data containing time information

## Returns

true if synchronization succeeded, false if it failed

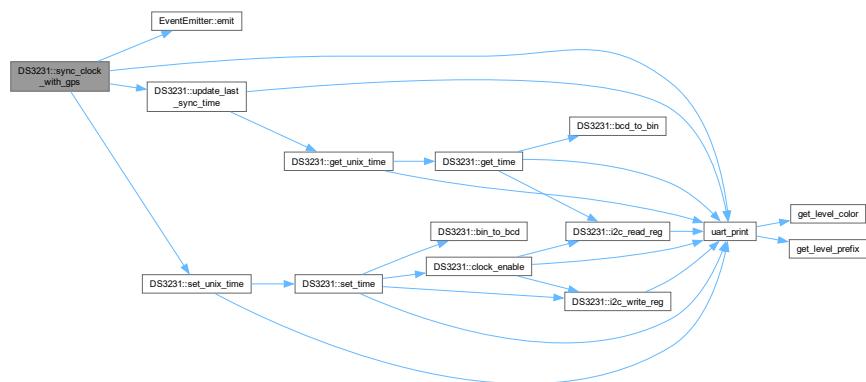
This method attempts to extract valid time data from the provided NMEA data and use it to update the RTC. It performs validity checks on the GPS data before attempting synchronization. If successful, it updates the last sync time and emits a SYNCED event. If unsuccessful, it emits a SYNC\_FAILED event.

## Note

This function emits events to the [EventEmitter](#) system that can be monitored by other components of the system.

Definition at line 421 of file DS3231.cpp.

Here is the call graph for this function:



#### 7.7.3.16 i2c read req()

```
int DS3231::i2c_read_reg (
    uint8_t reg_addr,
    size_t length,
    uint8_t * data) [private]
```

Reads data from a specific register on the DS3231.

## Parameters

in	<i>reg_addr</i>	Register address to read from
in	<i>length</i>	Number of bytes to read
out	<i>data</i>	Buffer to store read data

### Returns

0 on success, -1 on failure

**Parameters**

in	<i>reg_addr</i>	Register address to read from
in	<i>length</i>	Number of bytes to read
out	<i>data</i>	Buffer to store read data

**Returns**

0 on success, -1 on failure

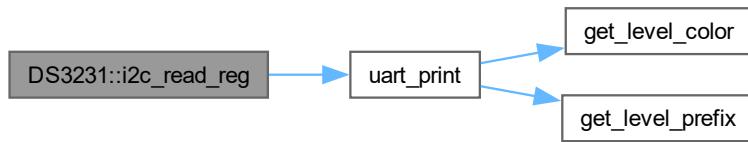
This method performs a thread-safe I<sup>2</sup>C read operation from the [DS3231](#). It first writes the register address to the device, then reads the requested number of bytes. All access is protected by a mutex to prevent concurrent I<sup>2</sup>C operations that could corrupt data.

**Note**

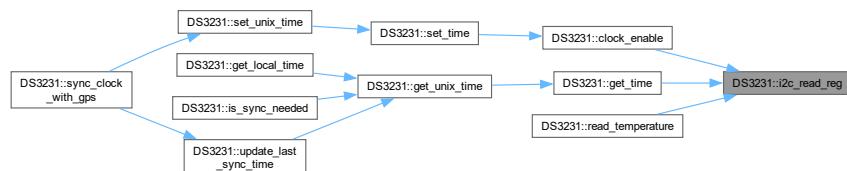
This is a low-level method used internally by the class.

Definition at line 467 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**7.7.3.17 i2c\_write\_reg()**

```

int DS3231::i2c_write_reg (
    uint8_t reg_addr,
    size_t length,
    uint8_t * data) [private]
  
```

Writes data to a specific register on the [DS3231](#).

**Parameters**

in	<i>reg_addr</i>	Register address to write to
in	<i>length</i>	Number of bytes to write
in	<i>data</i>	Buffer containing data to write

**Returns**

0 on success, -1 on failure

**Parameters**

in	<i>reg_addr</i>	Register address to write to
in	<i>length</i>	Number of bytes to write
in	<i>data</i>	Buffer containing data to write

**Returns**

0 on success, -1 on failure

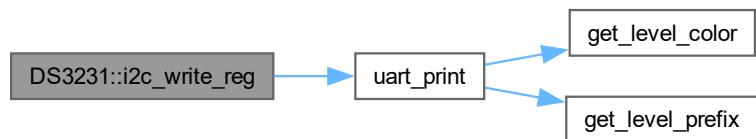
This method performs a thread-safe I<sup>2</sup>C write operation to the [DS3231](#). It combines the register address and data into a single buffer and sends it to the device. All access is protected by a mutex to prevent concurrent I<sup>2</sup>C operations that could corrupt data.

**Note**

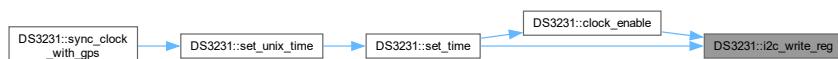
This is a low-level method used internally by the class.

Definition at line 508 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.7.3.18 bin\_to\_bcd()

```
uint8_t DS3231::bin_to_bcd (
    const uint8_t data) [private]
```

Converts binary value to BCD (Binary Coded Decimal)

Converts binary value to Binary-Coded Decimal (BCD) format.

#### Parameters

in	<i>data</i>	Binary value to convert (0-99)
----	-------------	--------------------------------

#### Returns

BCD representation of the input value

#### Parameters

in	<i>data</i>	Binary value to convert (0-99)
----	-------------	--------------------------------

#### Returns

BCD representation of the input value

The [DS3231](#) stores time values in BCD format where each nibble represents a decimal digit. This function converts a standard binary value to its BCD equivalent (e.g., 42 becomes 0x42).

Definition at line [538](#) of file [DS3231.cpp](#).

Here is the caller graph for this function:



### 7.7.3.19 bcd\_to\_bin()

```
uint8_t DS3231::bcd_to_bin (
    const uint8_t bcd) [private]
```

Converts BCD (Binary Coded Decimal) to binary value.

Converts Binary-Coded Decimal (BCD) to binary value.

**Parameters**

in	<i>bcd</i>	BCD value to convert
----	------------	----------------------

**Returns**

Binary representation of the input BCD value

**Parameters**

in	<i>bcd</i>	BCD value to convert
----	------------	----------------------

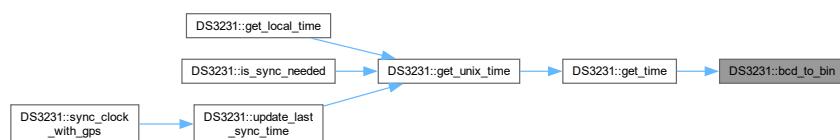
**Returns**

Binary representation of the input BCD value

The [DS3231](#) stores time values in BCD format where each nibble represents a decimal digit. This function converts a BCD value to its standard binary equivalent (e.g., 0x42 becomes 42).

Definition at line 554 of file [DS3231.cpp](#).

Here is the caller graph for this function:



## 7.7.4 Member Data Documentation

### 7.7.4.1 i2c

i2c\_inst\_t\* DS3231::i2c [private]

Pointer to I2C instance.

Definition at line 226 of file [DS3231.h](#).

### 7.7.4.2 ds3231\_addr

uint8\_t DS3231::ds3231\_addr [private]

I2C address of [DS3231](#).

Definition at line 227 of file [DS3231.h](#).

#### 7.7.4.3 `clock_mutex_`

```
recursive_mutex_t DS3231::clock_mutex_ [private]
```

Mutex for thread-safe I2C access.

Definition at line 265 of file [DS3231.h](#).

#### 7.7.4.4 `timezone_offset_minutes_`

```
int16_t DS3231::timezone_offset_minutes_ = 60 [private]
```

Timezone offset in minutes, default: UTC.

Definition at line 266 of file [DS3231.h](#).

#### 7.7.4.5 `sync_interval_minutes_`

```
uint32_t DS3231::sync_interval_minutes_ = 1440 [private]
```

Sync interval in minutes, default: 24 hours.

Definition at line 267 of file [DS3231.h](#).

#### 7.7.4.6 `last_sync_time_`

```
time_t DS3231::last_sync_time_ = 0 [private]
```

Last sync timestamp, 0 = never synced.

Definition at line 268 of file [DS3231.h](#).

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

- lib/clock/[DS3231.h](#)
- lib/clock/[DS3231.cpp](#)

## 7.8 `ds3231_data_t` Struct Reference

Structure to hold time and date information from [DS3231](#).

```
#include <DS3231.h>
```

## Public Attributes

- `uint8_t seconds`  
*Seconds (0-59)*
- `uint8_t minutes`  
*Minutes (0-59)*
- `uint8_t hours`  
*Hours (0-23)*
- `uint8_t day`  
*Day of the week (1-7)*
- `uint8_t date`  
*Date (1-31)*
- `uint8_t month`  
*Month (1-12)*
- `uint8_t year`  
*Year (0-99)*
- `bool century`  
*Century flag (0-1)*

### 7.8.1 Detailed Description

Structure to hold time and date information from [DS3231](#).

Definition at line [90](#) of file [DS3231.h](#).

### 7.8.2 Member Data Documentation

#### 7.8.2.1 seconds

```
uint8_t ds3231_data_t::seconds
```

Seconds (0-59)

Definition at line [91](#) of file [DS3231.h](#).

#### 7.8.2.2 minutes

```
uint8_t ds3231_data_t::minutes
```

Minutes (0-59)

Definition at line [92](#) of file [DS3231.h](#).

#### 7.8.2.3 hours

```
uint8_t ds3231_data_t::hours
```

Hours (0-23)

Definition at line [93](#) of file [DS3231.h](#).

### 7.8.2.4 day

```
uint8_t ds3231_data_t::day
```

Day of the week (1-7)

Definition at line 94 of file [DS3231.h](#).

### 7.8.2.5 date

```
uint8_t ds3231_data_t::date
```

Date (1-31)

Definition at line 95 of file [DS3231.h](#).

### 7.8.2.6 month

```
uint8_t ds3231_data_t::month
```

Month (1-12)

Definition at line 96 of file [DS3231.h](#).

### 7.8.2.7 year

```
uint8_t ds3231_data_t::year
```

Year (0-99)

Definition at line 97 of file [DS3231.h](#).

### 7.8.2.8 century

```
bool ds3231_data_t::century
```

Century flag (0-1)

Definition at line 98 of file [DS3231.h](#).

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

- lib/clock/[DS3231.h](#)

## 7.9 EventEmitter Class Reference

Provides a static method for emitting events.

```
#include <event_manager.h>
```

## Static Public Member Functions

- template<typename T>  
static void [emit](#) ([EventGroup group](#), T event)  
*Emits an event.*

### 7.9.1 Detailed Description

Provides a static method for emitting events.

Definition at line [317](#) of file [event\\_manager.h](#).

### 7.9.2 Member Function Documentation

#### 7.9.2.1 emit()

```
template<typename T>
static void EventEmitter::emit (
    EventGroup group,
    T event) [inline], [static]
```

Emits an event.

##### Parameters

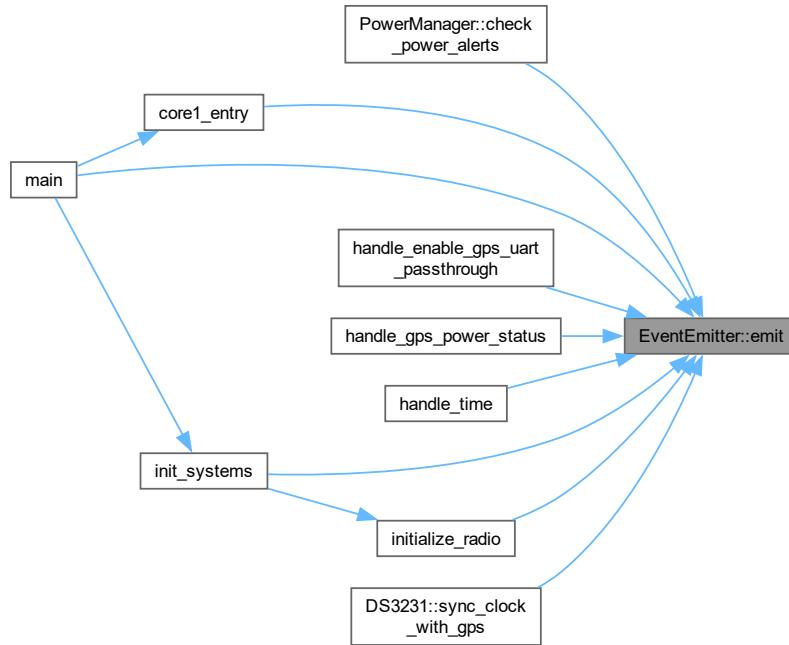
<i>group</i>	The event group.
<i>event</i>	The event identifier.

##### Template Parameters

<i>T</i>	The type of the event identifier.
----------	-----------------------------------

Definition at line [326](#) of file [event\\_manager.h](#).

Here is the caller graph for this function:



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

- lib/eventman/[event\\_manager.h](#)

## 7.10 EventLog Class Reference

Represents a single event log entry.

```
#include <event_manager.h>
```

### Public Member Functions

- std::string [to\\_string \(\) const](#)  
*Converts the [EventLog](#) to a string representation.*

### Public Attributes

- uint16\_t [id](#)  
*Sequence number.*
- uint32\_t [timestamp](#)  
*Unix timestamp or system time.*
- uint8\_t [group](#)  
*Event group identifier.*
- uint8\_t [event](#)  
*Specific event identifier.*

### 7.10.1 Detailed Description

Represents a single event log entry.

Definition at line 140 of file [event\\_manager.h](#).

### 7.10.2 Member Function Documentation

#### 7.10.2.1 `to_string()`

```
std::string EventLog::to_string () const [inline]
```

Converts the [EventLog](#) to a string representation.

**Returns**

A string representation of the [EventLog](#).

Definition at line 155 of file [event\\_manager.h](#).

Here is the caller graph for this function:



### 7.10.3 Member Data Documentation

#### 7.10.3.1 `id`

```
uint16_t EventLog::id
```

Sequence number.

Definition at line 143 of file [event\\_manager.h](#).

#### 7.10.3.2 `timestamp`

```
uint32_t EventLog::timestamp
```

Unix timestamp or system time.

Definition at line 145 of file [event\\_manager.h](#).

### 7.10.3.3 group

```
uint8_t EventLog::group
```

Event group identifier.

Definition at line 147 of file [event\\_manager.h](#).

### 7.10.3.4 event

```
uint8_t EventLog::event
```

Specific event identifier.

Definition at line 149 of file [event\\_manager.h](#).

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

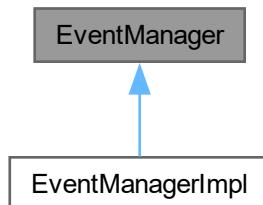
- lib/eventman/[event\\_manager.h](#)

## 7.11 EventManager Class Reference

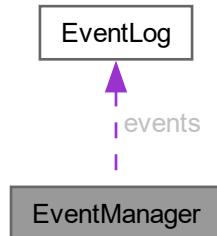
Manages the event logging system.

```
#include <event_manager.h>
```

Inheritance diagram for EventManager:



Collaboration diagram for EventManager:



## Public Member Functions

- `EventManager ()`  
`Constructor for the EventManager.`
- `virtual ~EventManager ()=default`  
`Virtual destructor for the EventManager.`
- `virtual void init ()`  
`Initializes the EventManager.`
- `void log_event (uint8_t group, uint8_t event)`  
`Logs an event.`
- `const EventLog & get_event (size_t index) const`  
`Retrieves an event from the event buffer.`
- `size_t get_event_count () const`  
`Gets the number of events in the buffer.`
- `virtual bool save_to_storage ()=0`  
`Saves the events to storage.`
- `virtual bool load_from_storage ()=0`  
`Loads the events from storage.`

## Protected Attributes

- `EventLog events [EVENT_BUFFER_SIZE]`  
`Event buffer.`
- `size_t eventCount`  
`Number of events in the buffer.`
- `size_t writeIndex`  
`Index of the next event to be written.`
- `mutex_t eventMutex`  
`Mutex for protecting the event buffer.`
- `size_t eventsSinceFlush`  
`Number of events since last flush to storage.`

## Static Protected Attributes

- `static uint16_t nextEventId = 0`  
`Static event ID counter.`

### 7.11.1 Detailed Description

Manages the event logging system.

Definition at line 169 of file `event_manager.h`.

### 7.11.2 Constructor & Destructor Documentation

#### 7.11.2.1 EventManager()

```
EventManager::EventManager () [inline]
```

Constructor for the `EventManager`.

Initializes the event buffer, mutex, and other internal variables.

Definition at line 175 of file `event_manager.h`.

### 7.11.2.2 ~EventManager()

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

Virtual destructor for the [EventManager](#).

## 7.11.3 Member Function Documentation

### 7.11.3.1 init()

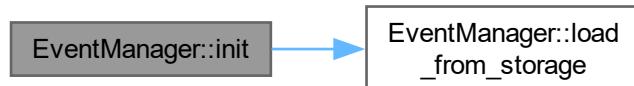
```
virtual void EventManager::init () [inline], [virtual]
```

Initializes the [EventManager](#).

Loads events from storage.

Definition at line 192 of file [event\\_manager.h](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.11.3.2 get\_event\_count()

```
size_t EventManager::get_event_count () const [inline]
```

Gets the number of events in the buffer.

#### Returns

The number of events in the buffer.

Definition at line 214 of file [event\\_manager.h](#).

### 7.11.3.3 `save_to_storage()`

```
virtual bool EventManager::save_to_storage () [pure virtual]
```

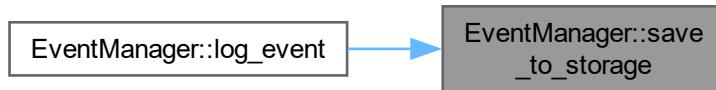
Saves the events to storage.

#### Returns

True if the events were successfully saved, false otherwise.

Implemented in [EventManagerImpl](#).

Here is the caller graph for this function:



### 7.11.3.4 `load_from_storage()`

```
virtual bool EventManager::load_from_storage () [pure virtual]
```

Loads the events from storage.

#### Returns

True if the events were successfully loaded, false otherwise.

Implemented in [EventManagerImpl](#).

Here is the caller graph for this function:



## 7.11.4 Member Data Documentation

### 7.11.4.1 `events`

```
EventLog EventManager::events [EVENT_BUFFER_SIZE] [protected]
```

Event buffer.

Definition at line 230 of file [event\\_manager.h](#).

#### 7.11.4.2 eventCount

```
size_t EventManager::eventCount [protected]
```

Number of events in the buffer.

Definition at line 232 of file [event\\_manager.h](#).

#### 7.11.4.3 writeIndex

```
size_t EventManager::writeIndex [protected]
```

Index of the next event to be written.

Definition at line 234 of file [event\\_manager.h](#).

#### 7.11.4.4 eventMutex

```
mutex_t EventManager::eventMutex [protected]
```

Mutex for protecting the event buffer.

Definition at line 236 of file [event\\_manager.h](#).

#### 7.11.4.5 nextEventId

```
uint16_t EventManager::nextEventId = 0 [static], [protected]
```

Static event ID counter.

Definition at line 238 of file [event\\_manager.h](#).

#### 7.11.4.6 eventsSinceFlush

```
size_t EventManager::eventsSinceFlush [protected]
```

Number of events since last flush to storage.

Definition at line 240 of file [event\\_manager.h](#).

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

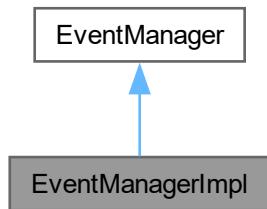
- lib/eventman/[event\\_manager.h](#)
- lib/eventman/[event\\_manager.cpp](#)

## 7.12 EventManagerImpl Class Reference

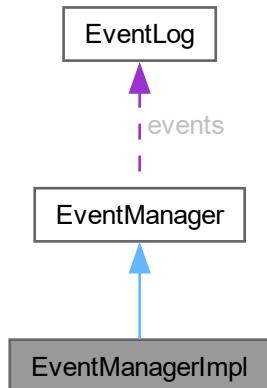
Implementation of the [EventManager](#) class.

```
#include <event_manager.h>
```

Inheritance diagram for EventManagerImpl:



Collaboration diagram for EventManagerImpl:



### Public Member Functions

- [EventManagerImpl \(\)](#)  
*Constructor for the `EventManagerImpl`.*
- bool [save\\_to\\_storage \(\)](#) override  
*Saves the events to storage.*
- bool [load\\_from\\_storage \(\)](#) override  
*Loads the events from storage.*

## Public Member Functions inherited from [EventManager](#)

- [EventManager \(\)](#)  
*Constructor for the [EventManager](#).*
- virtual [~EventManager \(\)=default](#)  
*Virtual destructor for the [EventManager](#).*
- virtual void [init \(\)](#)  
*Initializes the [EventManager](#).*
- void [log\\_event \(uint8\\_t group, uint8\\_t event\)](#)  
*Logs an event.*
- const [EventLog & get\\_event \(size\\_t index\) const](#)  
*Retrieves an event from the event buffer.*
- size\_t [get\\_event\\_count \(\) const](#)  
*Gets the number of events in the buffer.*

## Additional Inherited Members

## Protected Attributes inherited from [EventManager](#)

- [EventLog events \[EVENT\\_BUFFER\\_SIZE\]](#)  
*Event buffer.*
- size\_t [eventCount](#)  
*Number of events in the buffer.*
- size\_t [writeIndex](#)  
*Index of the next event to be written.*
- mutex\_t [eventMutex](#)  
*Mutex for protecting the event buffer.*
- size\_t [eventsSinceFlush](#)  
*Number of events since last flush to storage.*

## Static Protected Attributes inherited from [EventManager](#)

- static uint16\_t [nextEventId = 0](#)  
*Static event ID counter.*

### 7.12.1 Detailed Description

Implementation of the [EventManager](#) class.

Definition at line [248](#) of file [event\\_manager.h](#).

## 7.12.2 Constructor & Destructor Documentation

### 7.12.2.1 EventManagerImpl()

```
EventManagerImpl::EventManagerImpl () [inline]
```

Constructor for the [EventManagerImpl](#).

Initializes the [EventManagerImpl](#) and calls the init method.

Definition at line [254](#) of file [event\\_manager.h](#).

Here is the call graph for this function:



## 7.12.3 Member Function Documentation

### 7.12.3.1 save\_to\_storage()

```
bool EventManagerImpl::save_to_storage () [inline], [override], [virtual]
```

Saves the events to storage.

#### Returns

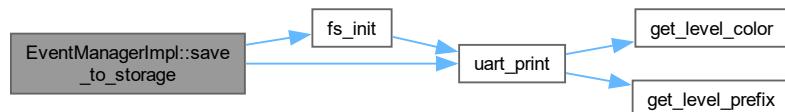
True if the events were successfully saved, false otherwise.

This method is not yet implemented.

Implements [EventManager](#).

Definition at line [264](#) of file [event\\_manager.h](#).

Here is the call graph for this function:



### 7.12.3.2 load\_from\_storage()

```
bool EventManagerImpl::load_from_storage () [inline], [override], [virtual]
```

Loads the events from storage.

#### Returns

True if the events were successfully loaded, false otherwise.

This method is not yet implemented.

Implements [EventManager](#).

Definition at line 301 of file [event\\_manager.h](#).

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

- lib/eventman/[event\\_manager.h](#)

## 7.13 FileHandle Struct Reference

```
#include <storage.h>
```

#### Public Attributes

- int [fd](#)
- bool [is\\_open](#)

### 7.13.1 Detailed Description

Definition at line 19 of file [storage.h](#).

### 7.13.2 Member Data Documentation

#### 7.13.2.1 fd

```
int FileHandle::fd
```

Definition at line 20 of file [storage.h](#).

#### 7.13.2.2 is\_open

```
bool FileHandle::is_open
```

Definition at line 21 of file [storage.h](#).

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

- lib/storage/[storage.h](#)

## 7.14 Frame Struct Reference

Represents a communication frame used for data exchange.

```
#include <protocol.h>
```

### Public Attributes

- std::string `header`
- uint8\_t `direction`
- OperationType `operationType`
- uint8\_t `group`
- uint8\_t `command`
- std::string `value`
- std::string `unit`
- std::string `footer`

### 7.14.1 Detailed Description

Represents a communication frame used for data exchange.

This structure encapsulates the different components of a communication frame, including the header, direction, operation type, group ID, command ID, payload value, unit, and footer. It is used for both encoding and decoding messages.

#### Note

The `header` and `footer` fields are used to mark the beginning and end of the frame, respectively.

The `direction` field indicates the direction of the communication (0 = ground->sat, 1 = sat->ground).

The `operationType` field specifies the type of operation being performed (e.g., GET, SET, ANS, ERR, INF).

The `group` and `command` fields identify the specific command being executed.

The `value` field contains the payload data.

The `unit` field specifies the unit of measurement for the payload data.

#### Example Usage:

```
// Creating a Frame instance
Frame myFrame;
myFrame.header = FRAME_BEGIN;
myFrame.direction = 1;
myFrame.operationType = OperationType::ANS;
myFrame.group = 2;
myFrame.command = 5;
myFrame.value = "25.5";
myFrame.unit = "VOLT";
myFrame.footer = FRAME_END;

// Encoding the Frame to a string
std::string encodedFrame = frame_encode(myFrame);
```

#### Example Instances:

```
// Example of a GET command
Frame getCommand;
getCommand.header = FRAME_BEGIN;
getCommand.direction = 0;
getCommand.operationType = OperationType::GET;
getCommand.group = 1;
getCommand.command = 10;
```

```
getCommand.value = "";
getCommand.unit = "";
getCommand.footer = FRAME_END;

// Example of an ANSWER command
Frame answerCommand;
answerCommand.header = FRAME_BEGIN;
answerCommand.direction = 1;
answerCommand.operationType = OperationType::ANS;
answerCommand.group = 1;
answerCommand.command = 10;
answerCommand.value = "OK";
answerCommand.unit = "";
answerCommand.footer = FRAME_END;
```

#### Example of Encoded Frames:

```
// Encoded GET command example:
// KBST;0;GET;1;10;;TSBK

// Encoded SET command example:
// KBST;0;SET;2;5;25.5;VOLT;TSBK

// Encoded ANSWER command example:
// KBST;1;ANS;1;10;OK;;TSBK

// Encoded ERROR command example:
// KBST;1;ERR;3;1;Invalid Parameter;;TSBK

// Encoded INFO command example:
// KBST;1;INF;4;2;System Booted;;TSBK
```

Definition at line 227 of file [protocol.h](#).

## 7.14.2 Member Data Documentation

### 7.14.2.1 header

```
std::string Frame::header
```

Definition at line 228 of file [protocol.h](#).

### 7.14.2.2 direction

```
uint8_t Frame::direction
```

Definition at line 229 of file [protocol.h](#).

### 7.14.2.3 operationType

```
OperationType Frame::operationType
```

Definition at line 230 of file [protocol.h](#).

### 7.14.2.4 group

```
uint8_t Frame::group
```

Definition at line 231 of file [protocol.h](#).

#### 7.14.2.5 command

```
uint8_t Frame::command
```

Definition at line 232 of file [protocol.h](#).

#### 7.14.2.6 value

```
std::string Frame::value
```

Definition at line 233 of file [protocol.h](#).

#### 7.14.2.7 unit

```
std::string Frame::unit
```

Definition at line 234 of file [protocol.h](#).

#### 7.14.2.8 footer

```
std::string Frame::footer
```

Definition at line 235 of file [protocol.h](#).

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

- lib/comms/[protocol.h](#)

## 7.15 HMC5883L Class Reference

```
#include <HMC5883L.h>
```

### Public Member Functions

- [HMC5883L](#) (*i2c\_inst\_t* \**i2c*, *uint8\_t* **address**=0x0D)
- bool [init](#) ()
- bool [read](#) (*int16\_t* &*x*, *int16\_t* &*y*, *int16\_t* &*z*)

### Private Member Functions

- bool [write\\_register](#) (*uint8\_t* *reg*, *uint8\_t* *value*)
- bool [read\\_register](#) (*uint8\_t* *reg*, *uint8\_t* \**buffer*, *size\_t* *length*)

### Private Attributes

- i2c\_inst\_t \* [i2c](#)
- uint8\_t [address](#)

### 7.15.1 Detailed Description

Definition at line [6](#) of file [HMC5883L.h](#).

### 7.15.2 Constructor & Destructor Documentation

#### 7.15.2.1 HMC5883L()

```
HMC5883L::HMC5883L (
    i2c_inst_t * i2c,
    uint8_t address = 0x0D)
```

Definition at line [3](#) of file [HMC5883L.cpp](#).

### 7.15.3 Member Function Documentation

#### 7.15.3.1 init()

```
bool HMC5883L::init ()
```

Definition at line [5](#) of file [HMC5883L.cpp](#).

Here is the call graph for this function:



### 7.15.3.2 `read()`

```
bool HMC5883L::read (
    int16_t & x,
    int16_t & y,
    int16_t & z)
```

Definition at line 13 of file [HMC5883L.cpp](#).

Here is the call graph for this function:



### 7.15.3.3 `write_register()`

```
bool HMC5883L::write_register (
    uint8_t reg,
    uint8_t value) [private]
```

Definition at line 28 of file [HMC5883L.cpp](#).

Here is the caller graph for this function:



### 7.15.3.4 `read_register()`

```
bool HMC5883L::read_register (
    uint8_t reg,
    uint8_t * buffer,
    size_t length) [private]
```

Definition at line 33 of file [HMC5883L.cpp](#).

Here is the caller graph for this function:



## 7.15.4 Member Data Documentation

### 7.15.4.1 i2c

```
i2c_inst_t* HMC5883L::i2c [private]
```

Definition at line 13 of file [HMC5883L.h](#).

### 7.15.4.2 address

```
uint8_t HMC5883L::address [private]
```

Definition at line 14 of file [HMC5883L.h](#).

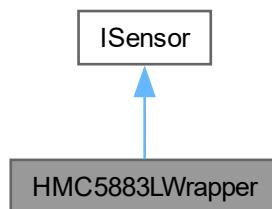
The documentation for this class was generated from the following files:

- lib/sensors/HMC5883L/[HMC5883L.h](#)
- lib/sensors/HMC5883L/[HMC5883L.cpp](#)

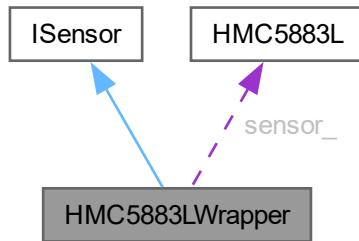
## 7.16 HMC5883LWrapper Class Reference

```
#include <HMC5883L_WRAPPER.h>
```

Inheritance diagram for HMC5883LWrapper:



Collaboration diagram for HMC5883LWrapper:



## Public Member Functions

- `HMC5883LWrapper (i2c_inst_t *i2c)`
- `bool init () override`
- `float read_data (SensorDataTypelIdentifier type) override`
- `bool is_initialized () const override`
- `SensorType get_type () const override`
- `bool configure (const std::map< std::string, std::string > &config) override`
- `uint8_t get_address () const override`

## Public Member Functions inherited from [ISensor](#)

- `virtual ~ISensor ()=default`

## Private Attributes

- `HMC5883L sensor_`
- `bool initialized_`

## 7.16.1 Detailed Description

Definition at line 7 of file [HMC5883L\\_WRAPPER.h](#).

## 7.16.2 Constructor & Destructor Documentation

### 7.16.2.1 `HMC5883LWrapper()`

```
HMC5883LWrapper::HMC5883LWrapper (
    i2c_inst_t * i2c)
```

Definition at line 5 of file [HMC5883L\\_WRAPPER.cpp](#).

### 7.16.3 Member Function Documentation

#### 7.16.3.1 init()

```
bool HMC5883LWrapper::init () [override], [virtual]
```

Implements [ISensor](#).

Definition at line [7](#) of file [HMC5883L\\_WRAPPER.cpp](#).

#### 7.16.3.2 read\_data()

```
float HMC5883LWrapper::read_data (
    SensorDataTypeIdentifier type) [override], [virtual]
```

Implements [ISensor](#).

Definition at line [12](#) of file [HMC5883L\\_WRAPPER.cpp](#).

#### 7.16.3.3 is\_initialized()

```
bool HMC5883LWrapper::is_initialized () const [override], [virtual]
```

Implements [ISensor](#).

Definition at line [35](#) of file [HMC5883L\\_WRAPPER.cpp](#).

#### 7.16.3.4 get\_type()

```
SensorType HMC5883LWrapper::get_type () const [override], [virtual]
```

Implements [ISensor](#).

Definition at line [39](#) of file [HMC5883L\\_WRAPPER.cpp](#).

#### 7.16.3.5 configure()

```
bool HMC5883LWrapper::configure (
    const std::map< std::string, std::string > & config) [override], [virtual]
```

Implements [ISensor](#).

Definition at line [43](#) of file [HMC5883L\\_WRAPPER.cpp](#).

#### 7.16.3.6 get\_address()

```
uint8_t HMC5883LWrapper::get_address () const [inline], [override], [virtual]
```

Implements [ISensor](#).

Definition at line [16](#) of file [HMC5883L\\_WRAPPER.h](#).

## 7.16.4 Member Data Documentation

### 7.16.4.1 `sensor_`

`HMC5883L HMC5883LWrapper::sensor_ [private]`

Definition at line 21 of file [HMC5883L\\_WRAPPER.h](#).

### 7.16.4.2 `initialized_`

`bool HMC5883LWrapper::initialized_ [private]`

Definition at line 22 of file [HMC5883L\\_WRAPPER.h](#).

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

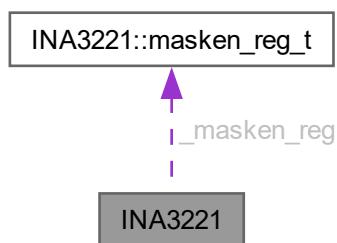
- lib/sensors/HMC5883L/[HMC5883L\\_WRAPPER.h](#)
- lib/sensors/HMC5883L/[HMC5883L\\_WRAPPER.cpp](#)

## 7.17 INA3221 Class Reference

[INA3221](#) Triple-Channel Power Monitor driver class.

#include <INA3221.h>

Collaboration diagram for INA3221:



## Classes

- struct `conf_reg_t`  
*Configuration register bit fields.*
- struct `masken_reg_t`  
*Mask/Enable register bit fields.*

## Public Member Functions

- `INA3221 (ina3221_addr_t addr, i2c_inst_t *i2c)`  
*Constructor for `INA3221` class.*
- `bool begin ()`  
*Initialize the `INA3221` device.*
- `uint16_t read_register (ina3221_reg_t reg)`  
*Read a register from the device.*
- `void reset ()`  
*Reset the `INA3221` to default settings.*
- `void set_mode_power_down ()`  
*Set device to power-down mode.*
- `void set_mode_continuous ()`  
*Set device to continuous measurement mode.*
- `void set_mode_triggered ()`  
*Set device to triggered measurement mode.*
- `void set_shunt_measurement_enable ()`  
*Enable shunt voltage measurements.*
- `void set_shunt_measurement_disable ()`  
*Disable shunt voltage measurements.*
- `void set_bus_measurement_enable ()`  
*Enable bus voltage measurements.*
- `void set_bus_measurement_disable ()`  
*Disable bus voltage measurements.*
- `void set_averaging_mode (ina3221_avg_mode_t mode)`  
*Set the averaging mode for measurements.*
- `void set_bus_conversion_time (ina3221_conv_time_t convTime)`  
*Set bus voltage conversion time.*
- `void set_shunt_conversion_time (ina3221_conv_time_t convTime)`  
*Set shunt voltage conversion time.*
- `uint16_t get_manufacturer_id ()`  
*Get the manufacturer ID of the device.*
- `uint16_t get_die_id ()`  
*Get the die ID of the device.*
- `int32_t get_shunt_voltage (ina3221_ch_t channel)`  
*Get shunt voltage for a specific channel.*
- `float get_current (ina3221_ch_t channel)`
- `float get_current_ma (ina3221_ch_t channel)`  
*Get current for a specific channel.*
- `float get_voltage (ina3221_ch_t channel)`  
*Get bus voltage for a specific channel.*
- `void set_warn_alert_limit (ina3221_ch_t channel, float voltage_v)`  
*Set warning alert voltage threshold for a channel.*
- `void set_crit_alert_limit (ina3221_ch_t channel, float voltage_v)`  
*Set critical alert voltage threshold for a channel.*
- `void set_power_valid_limit (float voltage_upper_v, float voltage_lower_v)`  
*Set power valid voltage range.*
- `void enable_alerts ()`  
*Enable all alert functions.*
- `bool get_warn_alert (ina3221_ch_t channel)`  
*Get warning alert status for a channel.*

- bool `get_crit_alert (ina3221_ch_t channel)`  
*Get critical alert status for a channel.*
- bool `get_power_valid_alert ()`  
*Get power valid alert status.*
- void `set_alert_latch (bool enable)`  
*Set alert latch mode.*

### Private Member Functions

- void `_read (ina3221_reg_t reg, uint16_t *val)`  
*Read a 16-bit register from the device.*
- void `_write (ina3221_reg_t reg, uint16_t *val)`  
*Write a 16-bit value to a register.*

### Private Attributes

- `i2c_inst_t * _i2c`
- `ina3221_addr_t _i2c_addr`
- `uint32_t _shuntRes [INA3221_CH_NUM]`
- `uint32_t _filterRes [INA3221_CH_NUM]`
- `masken_reg_t _masken_reg`

## 7.17.1 Detailed Description

[INA3221](#) Triple-Channel Power Monitor driver class.

Provides functionality for voltage, current, and power monitoring with configurable alerts and power valid monitoring

Definition at line 96 of file [INA3221.h](#).

## 7.17.2 Member Function Documentation

### 7.17.2.1 `_read()`

```
void INA3221::_read (
    ina3221_reg_t reg,
    uint16_t * val)  [private]
```

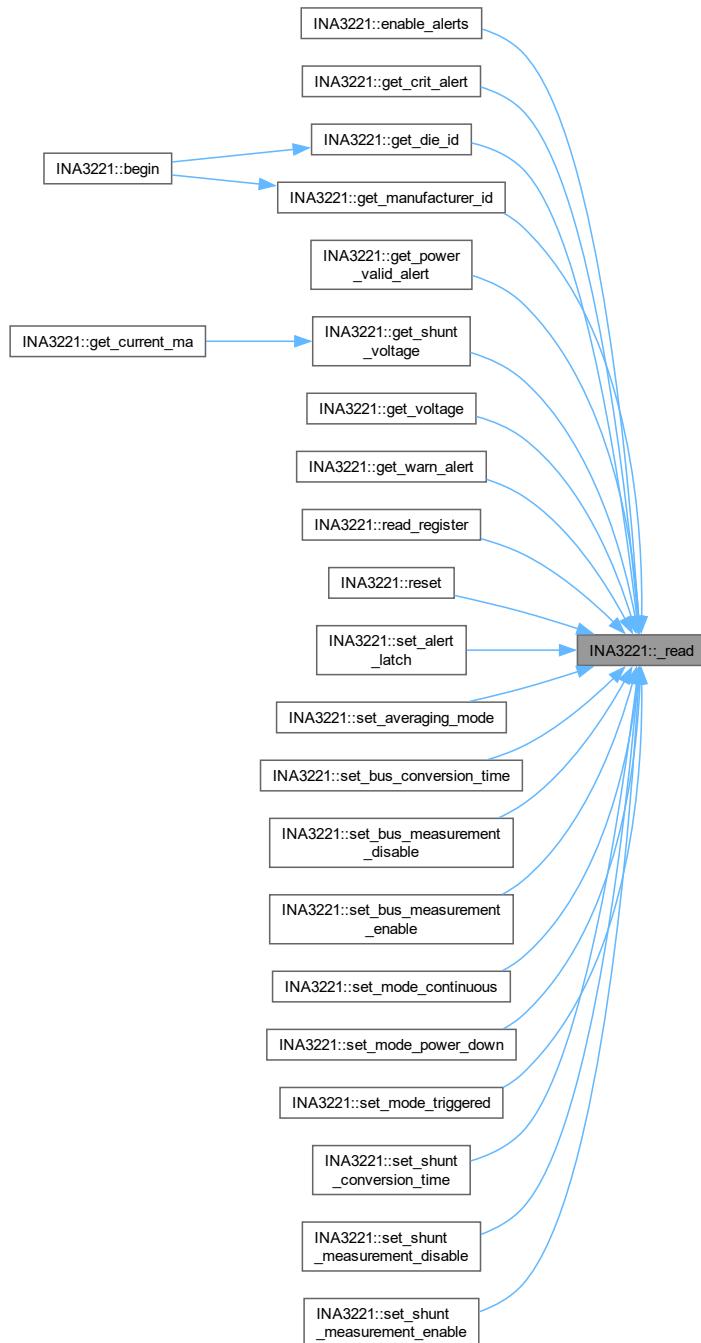
Read a 16-bit register from the device.

#### Parameters

<code>reg</code>	Register address
<code>val</code>	Pointer to store the read value

Definition at line 513 of file [INA3221.cpp](#).

Here is the caller graph for this function:



### 7.17.2.2 `_write()`

```
void INA3221::_write (
    ina3221_reg_t reg,
    uint16_t * val) [private]
```

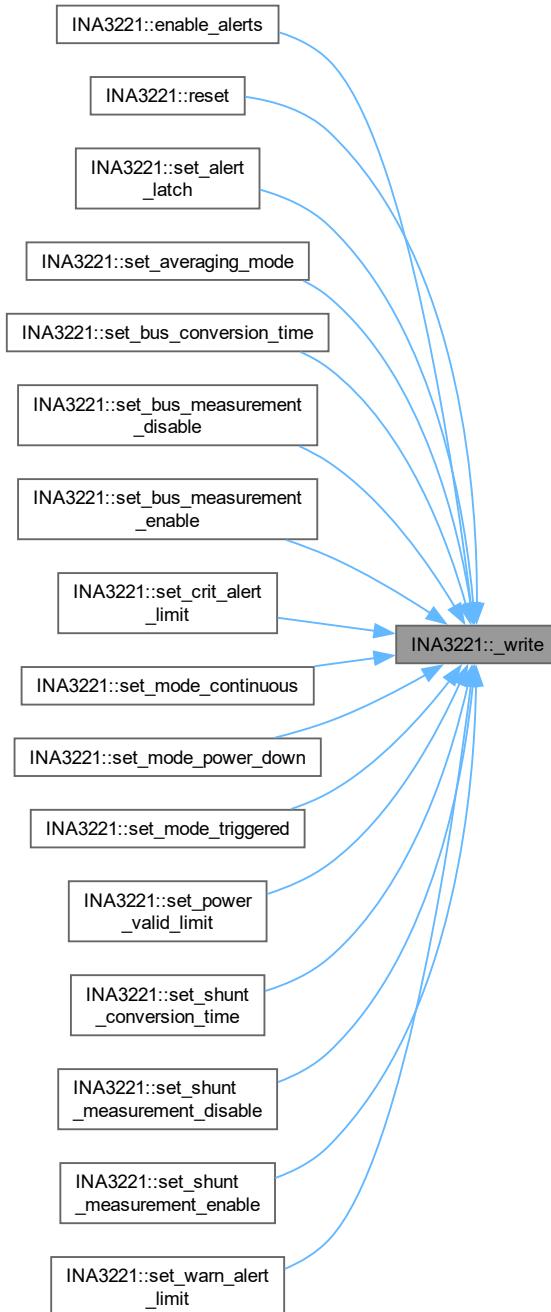
Write a 16-bit value to a register.

#### Parameters

<i>reg</i>	Register address
<i>val</i>	Pointer to the value to write

Definition at line 539 of file [INA3221.cpp](#).

Here is the caller graph for this function:



### 7.17.2.3 get\_current()

```
float INA3221::get_current (
    ina3221_ch_t channel)
```

### 7.17.3 Member Data Documentation

#### 7.17.3.1 \_i2c

```
i2c_inst_t* INA3221::_i2c [private]
```

Definition at line 136 of file [INA3221.h](#).

#### 7.17.3.2 \_i2c\_addr

```
ina3221_addr_t INA3221::_i2c_addr [private]
```

Definition at line 138 of file [INA3221.h](#).

#### 7.17.3.3 \_shuntRes

```
uint32_t INA3221::_shuntRes[INA3221_CH_NUM] [private]
```

Definition at line 141 of file [INA3221.h](#).

#### 7.17.3.4 \_filterRes

```
uint32_t INA3221::_filterRes[INA3221_CH_NUM] [private]
```

Definition at line 144 of file [INA3221.h](#).

#### 7.17.3.5 \_masken\_reg

```
masken_reg_t INA3221::_masken_reg [private]
```

Definition at line 147 of file [INA3221.h](#).

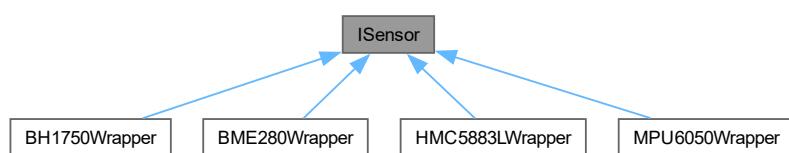
The documentation for this class was generated from the following files:

- lib/powerman/INA3221/[INA3221.h](#)
- lib/powerman/INA3221/[INA3221.cpp](#)

## 7.18 ISensor Class Reference

```
#include <ISensor.h>
```

Inheritance diagram for ISensor:



## Public Member Functions

- virtual `~ISensor()`=default
- virtual bool `init()`=0
- virtual float `read_data(SensorDataTypelIdentifier type)`=0
- virtual bool `is_initialized()` const =0
- virtual `SensorType get_type()` const =0
- virtual bool `configure(const std::map<std::string, std::string> &config)`=0
- virtual uint8\_t `get_address()` const =0

### 7.18.1 Detailed Description

Definition at line 34 of file [ISensor.h](#).

### 7.18.2 Constructor & Destructor Documentation

#### 7.18.2.1 `~ISensor()`

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

### 7.18.3 Member Function Documentation

#### 7.18.3.1 `init()`

```
virtual bool ISensor::init () [pure virtual]
```

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

#### 7.18.3.2 `read_data()`

```
virtual float ISensor::read_data (
    SensorDataTypelIdentifier type) [pure virtual]
```

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

#### 7.18.3.3 `is_initialized()`

```
virtual bool ISensor::is_initialized () const [pure virtual]
```

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

#### 7.18.3.4 `get_type()`

```
virtual SensorType ISensor::get_type () const [pure virtual]
```

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

### 7.18.3.5 `configure()`

```
virtual bool ISensor::configure (
    const std::map< std::string, std::string > & config) [pure virtual]
```

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

### 7.18.3.6 `get_address()`

```
virtual uint8_t ISensor::get_address () const [pure virtual]
```

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), and [HMC5883LWrapper](#).

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

- lib/sensors/[ISensor.h](#)

## 7.19 `INA3221::masken_reg_t` Struct Reference

Mask/Enable register bit fields.

### Public Attributes

- `uint16_t conv_ready:1`
- `uint16_t timing_ctrl_alert:1`
- `uint16_t pwr_valid_alert:1`
- `uint16_t warn_alert_ch3:1`
- `uint16_t warn_alert_ch2:1`
- `uint16_t warn_alert_ch1:1`
- `uint16_t shunt_sum_alert:1`
- `uint16_t crit_alert_ch3:1`
- `uint16_t crit_alert_ch2:1`
- `uint16_t crit_alert_ch1:1`
- `uint16_t crit_alert_latch_en:1`
- `uint16_t warn_alert_latch_en:1`
- `uint16_t shunt_sum_en_ch3:1`
- `uint16_t shunt_sum_en_ch2:1`
- `uint16_t shunt_sum_en_ch1:1`
- `uint16_t reserved:1`

### 7.19.1 Detailed Description

Mask/Enable register bit fields.

Definition at line 117 of file [INA3221.h](#).

## 7.19.2 Member Data Documentation

### 7.19.2.1 conv\_ready

```
uint16_t INA3221::masken_reg_t::conv_ready
```

Definition at line 118 of file [INA3221.h](#).

### 7.19.2.2 timing\_ctrl\_alert

```
uint16_t INA3221::masken_reg_t::timing_ctrl_alert
```

Definition at line 119 of file [INA3221.h](#).

### 7.19.2.3 pwr\_valid\_alert

```
uint16_t INA3221::masken_reg_t::pwr_valid_alert
```

Definition at line 120 of file [INA3221.h](#).

### 7.19.2.4 warn\_alert\_ch3

```
uint16_t INA3221::masken_reg_t::warn_alert_ch3
```

Definition at line 121 of file [INA3221.h](#).

### 7.19.2.5 warn\_alert\_ch2

```
uint16_t INA3221::masken_reg_t::warn_alert_ch2
```

Definition at line 122 of file [INA3221.h](#).

### 7.19.2.6 warn\_alert\_ch1

```
uint16_t INA3221::masken_reg_t::warn_alert_ch1
```

Definition at line 123 of file [INA3221.h](#).

### 7.19.2.7 shunt\_sum\_alert

```
uint16_t INA3221::masken_reg_t::shunt_sum_alert
```

Definition at line 124 of file [INA3221.h](#).

### 7.19.2.8 crit\_alert\_ch3

```
uint16_t INA3221::masken_reg_t::crit_alert_ch3
```

Definition at line 125 of file [INA3221.h](#).

### 7.19.2.9 crit\_alert\_ch2

```
uint16_t INA3221::masken_reg_t::crit_alert_ch2
```

Definition at line 126 of file [INA3221.h](#).

### 7.19.2.10 crit\_alert\_ch1

```
uint16_t INA3221::masken_reg_t::crit_alert_ch1
```

Definition at line 127 of file [INA3221.h](#).

### 7.19.2.11 crit\_alert\_latch\_en

```
uint16_t INA3221::masken_reg_t::crit_alert_latch_en
```

Definition at line 128 of file [INA3221.h](#).

### 7.19.2.12 warn\_alert\_latch\_en

```
uint16_t INA3221::masken_reg_t::warn_alert_latch_en
```

Definition at line 129 of file [INA3221.h](#).

### 7.19.2.13 shunt\_sum\_en\_ch3

```
uint16_t INA3221::masken_reg_t::shunt_sum_en_ch3
```

Definition at line 130 of file [INA3221.h](#).

### 7.19.2.14 shunt\_sum\_en\_ch2

```
uint16_t INA3221::masken_reg_t::shunt_sum_en_ch2
```

Definition at line 131 of file [INA3221.h](#).

### 7.19.2.15 shunt\_sum\_en\_ch1

```
uint16_t INA3221::masken_reg_t::shunt_sum_en_ch1
```

Definition at line 132 of file [INA3221.h](#).

### 7.19.2.16 reserved

```
uint16_t INA3221::masken_reg_t::reserved
```

Definition at line 133 of file [INA3221.h](#).

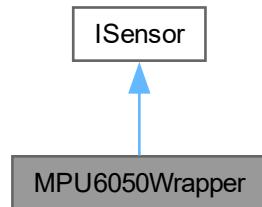
The documentation for this struct was generated from the following file:

- [lib/powerman/INA3221/INA3221.h](#)

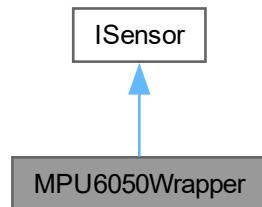
## 7.20 MPU6050Wrapper Class Reference

```
#include <MPU6050_WRAPPER.h>
```

Inheritance diagram for MPU6050Wrapper:



Collaboration diagram for MPU6050Wrapper:



### Public Member Functions

- [`MPU6050Wrapper \(\)`](#)
- [`bool init \(\) override`](#)
- [`float read\_data \(SensorDataTypelIdentifier type\) override`](#)
- [`bool is\_initialized \(\) const override`](#)
- [`SensorType get\_type \(\) const override`](#)
- [`bool configure \(const std::map< std::string, std::string > &config\)`](#)

## Public Member Functions inherited from [ISensor](#)

- virtual `~ISensor ()=default`
- virtual `uint8_t get_address () const =0`

## Private Attributes

- `MPU6050 sensor_`
- `bool initialized_ = false`

### 7.20.1 Detailed Description

Definition at line 9 of file [MPU6050\\_WRAPPER.h](#).

### 7.20.2 Constructor & Destructor Documentation

#### 7.20.2.1 `MPU6050Wrapper()`

```
MPU6050Wrapper::MPU6050Wrapper ()
```

### 7.20.3 Member Function Documentation

#### 7.20.3.1 `init()`

```
bool MPU6050Wrapper::init () [override], [virtual]
```

Implements [ISensor](#).

#### 7.20.3.2 `read_data()`

```
float MPU6050Wrapper::read_data (
    SensorDataTypeIdentifier type) [override], [virtual]
```

Implements [ISensor](#).

#### 7.20.3.3 `is_initialized()`

```
bool MPU6050Wrapper::is_initialized () const [override], [virtual]
```

Implements [ISensor](#).

#### 7.20.3.4 `get_type()`

```
SensorType MPU6050Wrapper::get_type () const [override], [virtual]
```

Implements [ISensor](#).

### 7.20.3.5 configure()

```
bool MPU6050Wrapper::configure (
    const std::map< std::string, std::string > & config) [virtual]
```

Implements [ISensor](#).

## 7.20.4 Member Data Documentation

### 7.20.4.1 sensor\_

```
MPU6050 MPU6050Wrapper::sensor_ [private]
```

Definition at line 11 of file [MPU6050\\_WRAPPER.h](#).

### 7.20.4.2 initialized\_

```
bool MPU6050Wrapper::initialized_ = false [private]
```

Definition at line 12 of file [MPU6050\\_WRAPPER.h](#).

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

- lib/sensors/MPU6050/[MPU6050\\_WRAPPER.h](#)

## 7.21 NMEAData Class Reference

```
#include <NMEA_data.h>
```

### Public Member Functions

- [NMEAData \(\)](#)
- void [update\\_rmc\\_tokens](#) (const std::vector< std::string > &tokens)
- void [update\\_gga\\_tokens](#) (const std::vector< std::string > &tokens)
- std::vector< std::string > [get\\_rmc\\_tokens](#) () const
- std::vector< std::string > [get\\_gga\\_tokens](#) () const
- bool [has\\_valid\\_time](#) () const
- time\_t [get\\_unix\\_time](#) () const

### Private Attributes

- std::vector< std::string > [rmc\\_tokens\\_](#)
- std::vector< std::string > [gga\\_tokens\\_](#)
- mutex\_t [rmc\\_mutex\\_](#)
- mutex\_t [gga\\_mutex\\_](#)

### 7.21.1 Detailed Description

Definition at line 10 of file [NMEA\\_data.h](#).

### 7.21.2 Constructor & Destructor Documentation

#### 7.21.2.1 NMEAData()

```
NMEAData::NMEAData ()
```

Definition at line 5 of file [NMEA\\_data.cpp](#).

### 7.21.3 Member Function Documentation

#### 7.21.3.1 update\_rmc\_tokens()

```
void NMEAData::update_rmc_tokens (
    const std::vector< std::string > & tokens)
```

Definition at line 10 of file [NMEA\\_data.cpp](#).

#### 7.21.3.2 update\_gga\_tokens()

```
void NMEAData::update_gga_tokens (
    const std::vector< std::string > & tokens)
```

Definition at line 16 of file [NMEA\\_data.cpp](#).

#### 7.21.3.3 get\_rmc\_tokens()

```
std::vector< std::string > NMEAData::get_rmc_tokens () const
```

Definition at line 22 of file [NMEA\\_data.cpp](#).

#### 7.21.3.4 get\_gga\_tokens()

```
std::vector< std::string > NMEAData::get_gga_tokens () const
```

Definition at line 29 of file [NMEA\\_data.cpp](#).

### 7.21.3.5 has\_valid\_time()

```
bool NMEAData::has_valid_time () const
```

Definition at line 36 of file [NMEA\\_data.cpp](#).

Here is the caller graph for this function:



### 7.21.3.6 get\_unix\_time()

```
time_t NMEAData::get_unix_time () const
```

Definition at line 40 of file [NMEA\\_data.cpp](#).

Here is the call graph for this function:



## 7.21.4 Member Data Documentation

### 7.21.4.1 rmc\_tokens\_

```
std::vector<std::string> NMEAData::rmc_tokens_ [private]
```

Definition at line 24 of file [NMEA\\_data.h](#).

### 7.21.4.2 gga\_tokens\_

```
std::vector<std::string> NMEAData::gga_tokens_ [private]
```

Definition at line 25 of file [NMEA\\_data.h](#).

#### 7.21.4.3 rmc\_mutex\_

```
mutex_t NMEAData::rmc_mutex_ [private]
```

Definition at line 26 of file [NMEA\\_data.h](#).

#### 7.21.4.4 gga\_mutex\_

```
mutex_t NMEAData::gga_mutex_ [private]
```

Definition at line 27 of file [NMEA\\_data.h](#).

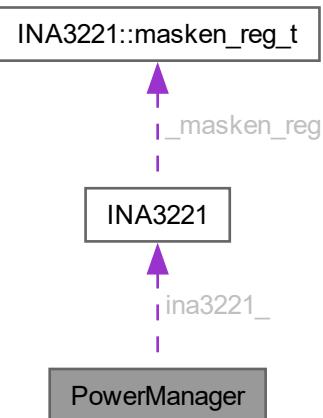
The documentation for this class was generated from the following files:

- lib/location/NMEA/[NMEA\\_data.h](#)
- lib/location/NMEA/[NMEA\\_data.cpp](#)

## 7.22 PowerManager Class Reference

```
#include <PowerManager.h>
```

Collaboration diagram for PowerManager:



## Public Member Functions

- `PowerManager (i2c_inst_t *i2c)`
- `bool initialize ()`
- `std::string read_device_ids ()`
- `float get_current_charge_solar ()`
- `float get_current_charge_usb ()`
- `float get_current_charge_total ()`
- `float get_current_draw ()`
- `float get_voltage_battery ()`
- `float get_voltage_5v ()`
- `void configure (const std::map< std::string, std::string > &config)`
- `bool is_charging_solar ()`
- `bool is_charging_usb ()`
- `bool check_power_alerts ()`

## Static Public Attributes

- `static constexpr float SOLAR_CURRENT_THRESHOLD = 50.0f`
- `static constexpr float USB_CURRENT_THRESHOLD = 50.0f`
- `static constexpr float VOLTAGE_LOW_THRESHOLD = 4.6f`
- `static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f`
- `static constexpr float FALL_RATE_THRESHOLD = -0.02f`
- `static constexpr int FALLING_TREND_REQUIRED = 3`

## Private Attributes

- `INA3221 ina3221_`
- `bool initialized_`
- `recursive_mutex_t powerman_mutex_`
- `bool charging_solar_active_ = false`
- `bool charging_usb_active_ = false`

### 7.22.1 Detailed Description

Definition at line 11 of file [PowerManager.h](#).

### 7.22.2 Constructor & Destructor Documentation

#### 7.22.2.1 PowerManager()

```
PowerManager::PowerManager (
    i2c_inst_t * i2c)
```

Definition at line 6 of file [PowerManager.cpp](#).

### 7.22.3 Member Function Documentation

#### 7.22.3.1 initialize()

```
bool PowerManager::initialize ()
```

Definition at line 11 of file [PowerManager.cpp](#).

#### 7.22.3.2 read\_device\_ids()

```
std::string PowerManager::read_device_ids ()
```

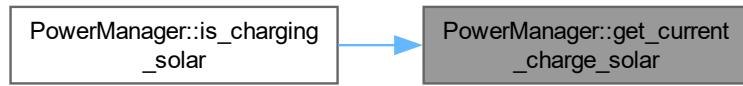
Definition at line 28 of file [PowerManager.cpp](#).

#### 7.22.3.3 get\_current\_charge\_solar()

```
float PowerManager::get_current_charge_solar ()
```

Definition at line 74 of file [PowerManager.cpp](#).

Here is the caller graph for this function:



#### 7.22.3.4 get\_current\_charge\_usb()

```
float PowerManager::get_current_charge_usb ()
```

Definition at line 58 of file [PowerManager.cpp](#).

Here is the caller graph for this function:



### 7.22.3.5 `get_current_charge_total()`

```
float PowerManager::get_current_charge_total ()
```

Definition at line 82 of file [PowerManager.cpp](#).

### 7.22.3.6 `get_current_draw()`

```
float PowerManager::get_current_draw ()
```

Definition at line 66 of file [PowerManager.cpp](#).

### 7.22.3.7 `get_voltage_battery()`

```
float PowerManager::get_voltage_battery ()
```

Definition at line 42 of file [PowerManager.cpp](#).

### 7.22.3.8 `get_voltage_5v()`

```
float PowerManager::get_voltage_5v ()
```

Definition at line 50 of file [PowerManager.cpp](#).

### 7.22.3.9 `configure()`

```
void PowerManager::configure (
    const std::map< std::string, std::string > & config)
```

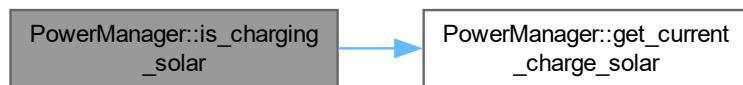
Definition at line 90 of file [PowerManager.cpp](#).

### 7.22.3.10 `is_charging_solar()`

```
bool PowerManager::is_charging_solar ()
```

Definition at line 119 of file [PowerManager.cpp](#).

Here is the call graph for this function:



### 7.22.3.11 `is_charging_usb()`

```
bool PowerManager::is_charging_usb ()
```

Definition at line 127 of file [PowerManager.cpp](#).

Here is the call graph for this function:



### 7.22.3.12 `check_power_alerts()`

```
bool PowerManager::check_power_alerts ()
```

Definition at line 135 of file [PowerManager.cpp](#).

Here is the call graph for this function:



## 7.22.4 Member Data Documentation

### 7.22.4.1 `SOLAR_CURRENT_THRESHOLD`

```
float PowerManager::SOLAR_CURRENT_THRESHOLD = 50.0f [static], [constexpr]
```

Definition at line 28 of file [PowerManager.h](#).

### 7.22.4.2 `USB_CURRENT_THRESHOLD`

```
float PowerManager::USB_CURRENT_THRESHOLD = 50.0f [static], [constexpr]
```

Definition at line 29 of file [PowerManager.h](#).

#### 7.22.4.3 VOLTAGE\_LOW\_THRESHOLD

```
float PowerManager::VOLTAGE_LOW_THRESHOLD = 4.6f [static], [constexpr]
```

Definition at line 30 of file [PowerManager.h](#).

#### 7.22.4.4 VOLTAGE\_OVERCHARGE\_THRESHOLD

```
float PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f [static], [constexpr]
```

Definition at line 31 of file [PowerManager.h](#).

#### 7.22.4.5 FALL\_RATE\_THRESHOLD

```
float PowerManager::FALL_RATE_THRESHOLD = -0.02f [static], [constexpr]
```

Definition at line 32 of file [PowerManager.h](#).

#### 7.22.4.6 FALLING\_TREND\_REQUIRED

```
int PowerManager::FALLING_TREND_REQUIRED = 3 [static], [constexpr]
```

Definition at line 33 of file [PowerManager.h](#).

#### 7.22.4.7 ina3221\_

```
INA3221 PowerManager::ina3221_ [private]
```

Definition at line 36 of file [PowerManager.h](#).

#### 7.22.4.8 initialized\_

```
bool PowerManager::initialized_ [private]
```

Definition at line 37 of file [PowerManager.h](#).

#### 7.22.4.9 powerman\_mutex\_

```
recursive_mutex_t PowerManager::powerman_mutex_ [private]
```

Definition at line 38 of file [PowerManager.h](#).

#### 7.22.4.10 charging\_solar\_active\_

```
bool PowerManager::charging_solar_active_ = false [private]
```

Definition at line 39 of file [PowerManager.h](#).

#### 7.22.4.11 charging\_usb\_active\_

```
bool PowerManager::charging_usb_active_ = false [private]
```

Definition at line 40 of file [PowerManager.h](#).

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

- lib/powerman/[PowerManager.h](#)
- lib/powerman/[PowerManager.cpp](#)

## 7.23 SensorWrapper Class Reference

Manages different sensor types and provides a unified interface for accessing sensor data.

```
#include <ISensor.h>
```

### Public Member Functions

- `bool sensor_init (SensorType type, i2c_inst_t *i2c=nullptr)`  
*Initializes a given sensor type on the specified I2C bus.*
- `bool sensor_configure (SensorType type, const std::map< std::string, std::string > &config)`  
*Configures an already initialized sensor with supplied settings.*
- `float sensor_read_data (SensorType sensorType, SensorDataTypelIdentifier dataType)`  
*Reads a specific data type (e.g., temperature, humidity) from a sensor.*
- `ISensor * get_sensor (SensorType type)`
- `std::vector< std::pair< SensorType, uint8_t > > scan_connected_sensors (i2c_inst_t *i2c)`  
*Scans the I2C bus for connected sensors.*
- `std::vector< std::pair< SensorType, uint8_t > > get_available_sensors ()`  
*Retrieves a list of available sensor types with their addresses.*

### Static Public Member Functions

- `static SensorWrapper & get_instance ()`  
*Provides a global instance of [SensorWrapper](#).*

### Private Member Functions

- `SensorWrapper ()`  
*Default constructor for [SensorWrapper](#).*

### Private Attributes

- `std::map< SensorType, ISensor * > sensors`

### 7.23.1 Detailed Description

Manages different sensor types and provides a unified interface for accessing sensor data.

Definition at line 45 of file [ISensor.h](#).

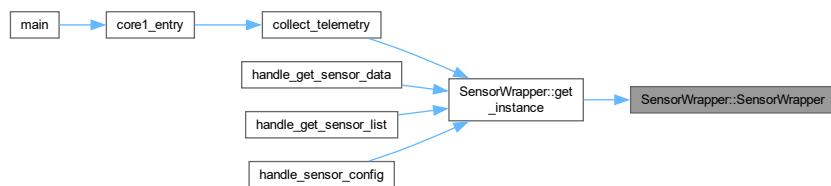
### 7.23.2 Constructor & Destructor Documentation

#### 7.23.2.1 SensorWrapper()

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

Default constructor for [SensorWrapper](#).

Here is the caller graph for this function:



### 7.23.3 Member Function Documentation

#### 7.23.3.1 get\_instance()

```
SensorWrapper & SensorWrapper::get_instance () [static]
```

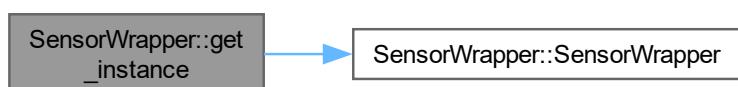
Provides a global instance of [SensorWrapper](#).

##### Returns

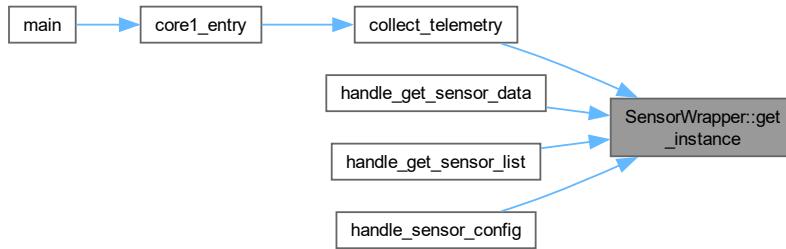
A reference to the single [SensorWrapper](#) instance.

Definition at line 24 of file [ISensor.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 7.23.3.2 sensor\_init()

```
bool SensorWrapper::sensor_init (
    SensorType type,
    i2c_inst_t * i2c = nullptr)
```

Initializes a given sensor type on the specified I2C bus.

#### Parameters

<i>type</i>	The sensor type (LIGHT, ENVIRONMENT, etc.).
<i>i2c</i>	The I2C interface pointer.

#### Returns

True if initialization succeeded, otherwise false.

Definition at line 42 of file [ISensor.cpp](#).

### 7.23.3.3 sensor\_configure()

```
bool SensorWrapper::sensor_configure (
    SensorType type,
    const std::map< std::string, std::string > & config)
```

Configures an already initialized sensor with supplied settings.

#### Parameters

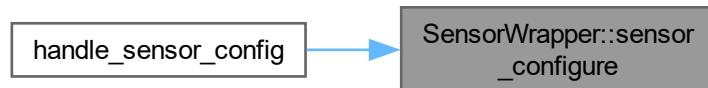
<i>type</i>	The sensor type.
<i>config</i>	Key-value pairs for sensor configuration.

**Returns**

True if the sensor was successfully configured, otherwise false.

Definition at line 67 of file [ISensor.cpp](#).

Here is the caller graph for this function:



#### 7.23.3.4 sensor\_read\_data()

```
float SensorWrapper::sensor_read_data (
    SensorType sensorType,
    SensorDataTypeIdentifier dataType)
```

Reads a specific data type (e.g., temperature, humidity) from a sensor.

**Parameters**

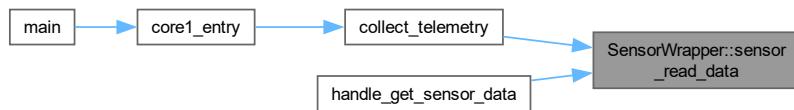
<code>sensorType</code>	The sensor type.
<code>dataType</code>	The type of data to read (light level, temperature, etc.).

**Returns**

The requested measurement. Returns 0.0f if sensor not found or uninitialized.

Definition at line 83 of file [ISensor.cpp](#).

Here is the caller graph for this function:



### 7.23.3.5 get\_sensor()

```
ISensor * SensorWrapper::get_sensor (
    SensorType type)
```

### 7.23.3.6 scan\_connected\_sensors()

```
std::vector< std::pair< SensorType, uint8_t > > SensorWrapper::scan_connected_sensors (
    i2c_inst_t * i2c)
```

Scans the I2C bus for connected sensors.

#### Parameters

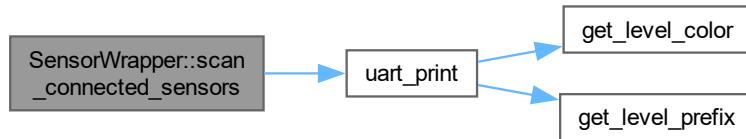
<i>i2c</i>	The I2C interface to scan
------------	---------------------------

#### Returns

A vector of pairs containing sensor type and I2C address of detected sensors

Definition at line 114 of file [ISensor.cpp](#).

Here is the call graph for this function:



### 7.23.3.7 get\_available\_sensors()

```
std::vector< std::pair< SensorType, uint8_t > > SensorWrapper::get_available_sensors ()
```

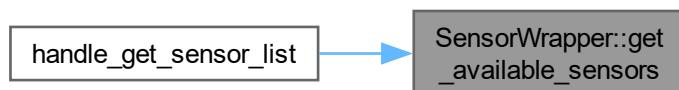
Retrieves a list of available sensor types with their addresses.

#### Returns

A vector of pairs containing sensor type and I2C address

Definition at line 96 of file [ISensor.cpp](#).

Here is the caller graph for this function:



### 7.23.4 Member Data Documentation

#### 7.23.4.1 sensors

```
std::map<SensorType, ISensor*> SensorWrapper::sensors [private]
```

Definition at line 57 of file [ISensor.h](#).

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

- lib/sensors/[ISensor.h](#)
- lib/sensors/[ISensor.cpp](#)

## 7.24 TelemetryRecord Struct Reference

Structure representing a single telemetry data point.

### Public Member Functions

- std::string [to\\_csv](#) () const

### Public Attributes

- uint32\_t [timestamp](#)
- float [battery\\_voltage](#)
- float [system\\_voltage](#)
- float [charge\\_current\\_usb](#)
- float [charge\\_current\\_solar](#)
- float [discharge\\_current](#)
- float [temperature](#)
- float [pressure](#)
- float [humidity](#)
- float [light\\_level](#)
- std::string [time](#)
- std::string [latitude](#)
- std::string [lat\\_dir](#)
- std::string [longitude](#)
- std::string [lon\\_dir](#)
- std::string [speed](#)
- std::string [course](#)
- std::string [date](#)
- std::string [fix\\_quality](#)
- std::string [satellites](#)
- std::string [altitude](#)

### 7.24.1 Detailed Description

Structure representing a single telemetry data point.

Contains all measurements from power subsystem, sensors, and GPS data collected at a specific point in time

Definition at line 57 of file [telemetry\\_manager.cpp](#).

### 7.24.2 Member Function Documentation

#### 7.24.2.1 `to_csv()`

```
std::string TelemetryRecord::to_csv () const [inline]
```

Definition at line 90 of file [telemetry\\_manager.cpp](#).

### 7.24.3 Member Data Documentation

#### 7.24.3.1 `timestamp`

```
uint32_t TelemetryRecord::timestamp
```

Unix timestamp of the record

Definition at line 58 of file [telemetry\\_manager.cpp](#).

#### 7.24.3.2 `battery_voltage`

```
float TelemetryRecord::battery_voltage
```

Battery voltage in volts

Definition at line 61 of file [telemetry\\_manager.cpp](#).

#### 7.24.3.3 `system_voltage`

```
float TelemetryRecord::system_voltage
```

System 5V rail voltage in volts

Definition at line 62 of file [telemetry\\_manager.cpp](#).

#### 7.24.3.4 `charge_current_usb`

```
float TelemetryRecord::charge_current_usb
```

USB charging current in mA

Definition at line 63 of file [telemetry\\_manager.cpp](#).

### 7.24.3.5 charge\_current\_solar

```
float TelemetryRecord::charge_current_solar
```

Solar charging current in mA

Definition at line [64](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.6 discharge\_current

```
float TelemetryRecord::discharge_current
```

Battery discharge current in mA

Definition at line [65](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.7 temperature

```
float TelemetryRecord::temperature
```

Temperature in degrees Celsius

Definition at line [68](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.8 pressure

```
float TelemetryRecord::pressure
```

Atmospheric pressure in hPa

Definition at line [69](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.9 humidity

```
float TelemetryRecord::humidity
```

Relative humidity in percent

Definition at line [70](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.10 light\_level

```
float TelemetryRecord::light_level
```

Light level in lux

Definition at line [71](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.11 time

```
std::string TelemetryRecord::time
```

UTC time from GPS

Definition at line [74](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.12 latitude

```
std::string TelemetryRecord::latitude
```

Latitude from GPS

Definition at line [75](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.13 lat\_dir

```
std::string TelemetryRecord::lat_dir
```

N/S latitude direction

Definition at line [76](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.14 longitude

```
std::string TelemetryRecord::longitude
```

Longitude from GPS

Definition at line [77](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.15 lon\_dir

```
std::string TelemetryRecord::lon_dir
```

E/W longitude direction

Definition at line [78](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.16 speed

```
std::string TelemetryRecord::speed
```

Speed in knots

Definition at line [79](#) of file [telemetry\\_manager.cpp](#).

### 7.24.3.17 course

```
std::string TelemetryRecord::course
```

Course in degrees

Definition at line 80 of file [telemetry\\_manager.cpp](#).

### 7.24.3.18 date

```
std::string TelemetryRecord::date
```

Date from GPS

Definition at line 81 of file [telemetry\\_manager.cpp](#).

### 7.24.3.19 fix\_quality

```
std::string TelemetryRecord::fix_quality
```

GPS fix quality

Definition at line 84 of file [telemetry\\_manager.cpp](#).

### 7.24.3.20 satellites

```
std::string TelemetryRecord::satellites
```

Number of satellites in view

Definition at line 85 of file [telemetry\\_manager.cpp](#).

### 7.24.3.21 altitude

```
std::string TelemetryRecord::altitude
```

Altitude in meters

Definition at line 86 of file [telemetry\\_manager.cpp](#).

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

- lib/telemetry/[telemetry\\_manager.cpp](#)



# Chapter 8

# File Documentation

## 8.1 build\_number.h File Reference

This graph shows which files directly or indirectly include this file:



### Macros

- `#define BUILD_NUMBER 404`

#### 8.1.1 Macro Definition Documentation

##### 8.1.1.1 BUILD\_NUMBER

```
#define BUILD_NUMBER 404
```

Definition at line 6 of file [build\\_number.h](#).

## 8.2 build\_number.h

[Go to the documentation of this file.](#)

```
00001 //This file is automatically generated by build_number.cmake
00002
00003 #ifndef CMAKE_BUILD_NUMBER_HEADER
00004 #define CMAKE_BUILD_NUMBER_HEADER
00005
00006 #define BUILD_NUMBER 404
00007
00008 #endif
```

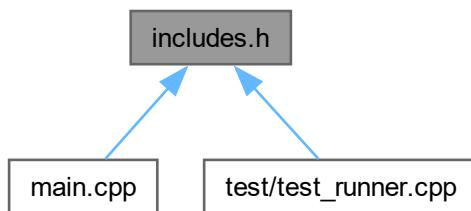
### 8.3 includes.h File Reference

```
#include <stdio.h>
#include "pico/stl.h"
#include "hardware/spi.h"
#include "hardware/i2c.h"
#include "hardware/uart.h"
#include "pico/multicore.h"
#include "event_manager.h"
#include "lib/powerman/PowerManager.h"
#include <pico/bootrom.h>
#include "ISensor.h"
#include "lib/sensors/BH1750/BH1750_WRAPPER.h"
#include "lib/sensors/BME280/BME280_WRAPPER.h"
#include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
#include "lib/sensors/MPU6050/MPU6050_WRAPPER.h"
#include "lib/clock/DS3231.h"
#include <iostream>
#include <iomanip>
#include <queue>
#include <chrono>
#include "protocol.h"
#include <atomic>
#include <map>
#include "pin_config.h"
#include "utils.h"
#include "communication.h"
#include "build_number.h"
#include "lib/location/gps_collector.h"
#include "lib/storage/storage.h"
#include "lib/storage/pico-vfs/include/filesystem/vfs.h"
#include "telemetry_manager.h"
```

Include dependency graph for includes.h:



This graph shows which files directly or indirectly include this file:



## 8.4 includes.h

[Go to the documentation of this file.](#)

```

00001 #ifndef INCLUDES_H
00002 #define INCLUDES_H
00003
00004 #include <stdio.h>
00005 #include "pico/stl.h"
00006 #include "hardware/spi.h"
00007 #include "hardware/i2c.h"
00008 #include "hardware/uart.h"
00009 #include "pico/multicore.h"
00010 #include "event_manager.h"
00011 #include "lib/powerman/PowerManager.h" // Corrected path
00012 #include <pico/bootrom.h>
00013
00014 #include "ISensor.h"
00015 #include "lib/sensors/BH1750/BH1750_WRAPPER.h" // Corrected path
00016 #include "lib/sensors/BME280/BME280_WRAPPER.h" // Corrected path
00017 #include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h" // Corrected path
00018 #include "lib/sensors/MPU6050/MPU6050_WRAPPER.h" // Corrected path
00019 #include "lib/clock/DS3231.h" // Corrected path
00020 #include <iostream>
00021 #include <iomanip>
00022 #include <queue>
00023 #include <chrono>
00024 #include "protocol.h"
00025 #include <atomic>
00026 #include <iostream>
00027 #include <map>
00028 #include "pin_config.h"
00029 #include "utils.h"
00030 #include "communication.h"
00031 #include "build_number.h"
00032 #include "lib/location/gps_collector.h"
00033 #include "lib/storage/storage.h" // Corrected path
00034 #include "lib/storage/pico-vfs/include/filesystem/vfs.h" // Corrected path
00035 #include "telemetry_manager.h"
00036
00037 #endif

```

## 8.5 lib/clock/DS3231.cpp File Reference

```

#include "DS3231.h"
#include "utils.h"
#include <stdio.h>
#include <mutex>
#include "event_manager.h"
#include "NMEA_data.h"

```

Include dependency graph for DS3231.cpp:



## 8.6 DS3231.cpp

[Go to the documentation of this file.](#)

```

00001 #include "DS3231.h"
00002 #include "utils.h"
00003 #include <stdio.h>
00004 #include <mutex>

```

```

00005 #include "event_manager.h"
00006 #include "NMEA_data.h"
00007
00016 DS3231::DS3231(i2c_inst_t *i2c_instance) : i2c(i2c_instance), ds3231_addr(DS3231_DEVICE_ADDRESS) {
00017     recursive_mutex_init(&clock_mutex);
00018 }
00019
00020
00033 int DS3231::set_time(ds3231_data_t *data) {
00034     uint8_t temp[7] = {0};
00035
00036     if (clock_enable() != 0) {
00037         uart_print("Failed to enable clock oscillator", VerbosityLevel::ERROR);
00038         return -1;
00039     }
00040
00041     if (data->seconds > 59)
00042         data->seconds = 59;
00043     if (data->minutes > 59)
00044         data->minutes = 59;
00045     if (data->hours > 23)
00046         data->hours = 23;
00047     if (data->day > 7)
00048         data->day = 7;
00049     else if (data->day < 1)
00050         data->day = 1;
00051     if (data->date > 31)
00052         data->date = 31;
00053     else if (data->date < 1)
00054         data->date = 1;
00055     if (data->month > 12)
00056         data->month = 12;
00057     else if (data->month < 1)
00058         data->month = 1;
00059     if (data->year > 99)
00060         data->year = 99;
00061
00062     temp[0] = bin_to_bcd(data->seconds);
00063     temp[1] = bin_to_bcd(data->minutes);
00064     temp[2] = bin_to_bcd(data->hours);
00065     temp[2] &= ~(0x01 « 6); // Clear 12/24 hour bit
00066     temp[3] = bin_to_bcd(data->day);
00067     temp[4] = bin_to_bcd(data->date);
00068     temp[5] = bin_to_bcd(data->month);
00069     if (data->century)
00070         temp[5] |= (0x01 « 7);
00071     temp[6] = bin_to_bcd(data->year);
00072
00073     std::string status = "BCD values to be written to DS3231: " + std::to_string(temp[0]) + " " +
00074             std::to_string(temp[1]) + " " + std::to_string(temp[2]) + " " +
00075             std::to_string(temp[3]) + " " + std::to_string(temp[4]) + " " +
00076             std::to_string(temp[5]) + " " + std::to_string(temp[6]);
00077
00078     uart_print(status, VerbosityLevel::DEBUG);
00079
00080     int result = i2c_write_reg(DS3231_SECONDS_REG, 7, temp);
00081     if (result != 0) {
00082         uart_print("i2c write failed", VerbosityLevel::ERROR);
00083         return -1;
00084     }
00085
00086     return 0;
00087 }
00088
00089
00102 int DS3231::get_time(ds3231_data_t *data) {
00103     std::string status;
00104     uint8_t raw_data[7];
00105     int result = i2c_read_reg(DS3231_SECONDS_REG, 7, raw_data);
00106     if (result != 0) {
00107         status = "Failed to read time from DS3231";
00108         uart_print(status, VerbosityLevel::ERROR);
00109         return -1;
00110     }
00111
00112     status = "Raw BCD values read from DS3231: " + std::to_string(raw_data[0]) + " " +
00113             std::to_string(raw_data[1]) + " " + std::to_string(raw_data[2]) + " " +
00114             std::to_string(raw_data[3]) + " " + std::to_string(raw_data[4]) + " " +
00115             std::to_string(raw_data[5]) + " " + std::to_string(raw_data[6]);
00116     uart_print(status, VerbosityLevel::DEBUG);
00117
00118     data->seconds = bcd_to_bin(raw_data[0] & 0x7F); // Masking for CH bit (clock halt)
00119     data->minutes = bcd_to_bin(raw_data[1] & 0x7F);
00120     data->hours = bcd_to_bin(raw_data[2] & 0x3F); // Masking for 12/24 hour mode bit
00121     data->day = raw_data[3] & 0x07; // Day of week (1-7)
00122     data->date = bcd_to_bin(raw_data[4] & 0x3F);
00123     data->month = bcd_to_bin(raw_data[5] & 0x1F); // Masking for century bit

```

```

00124     data->century = (raw_data[5] & 0x80) >> 7;
00125     data->year = bcd_to_bin(raw_data[6]);
00126
00127     if (data->seconds > 59 || data->minutes > 59 || data->hours > 23 ||
00128         data->day < 1 || data->day > 7 || data->date < 1 || data->date > 31 ||
00129         data->month < 1 || data->month > 12 || data->year > 99) {
00130         uart_print("Invalid data read from DS3231", VerbosityLevel::ERROR);
00131         return -1;
00132     }
00133
00134     uart_print("Reading time from DS3231", VerbosityLevel::DEBUG);
00135     std::string timeStr = "Time: " + std::to_string(data->hours) + ":" + std::to_string(data->minutes)
00136     + ":" + std::to_string(data->seconds);
00137     uart_print(timeStr, VerbosityLevel::DEBUG);
00138     std::string dateStr = "Date: " + std::to_string(data->date) + "/" + std::to_string(data->month) +
00139     "/" + std::to_string(data->year);
00140     uart_print(dateStr, VerbosityLevel::DEBUG);
00141
00142     return 0;
00143 }
00144
00145 int DS3231::read_temperature(float *resolution) {
00146     std::string status;
00147     uint8_t temp[2];
00148     int result = i2c_read_reg(DS3231_TEMPERATURE_MSB_REG, 2, temp);
00149     if (result != 0) {
00150         status = "Failed to read temperature from DS3231";
00151         uart_print(status, VerbosityLevel::ERROR);
00152         return -1;
00153     }
00154
00155     int8_t temperature_msb = (int8_t)temp[0];
00156     uint8_t temperature_lsb = temp[1] >> 6; // Only the 2 MSB are valid
00157
00158     *resolution = temperature_msb + (temperature_lsb * 0.25f); // 0.25 degree resolution
00159
00160     return 0;
00161 }
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184 int DS3231::set_unix_time(time_t unix_time) {
00185     struct tm *timeinfo = gmtime(&unix_time);
00186     if (timeinfo == NULL) {
00187         uart_print("Error: gmtime() failed", VerbosityLevel::ERROR);
00188         return -1;
00189     }
00190
00191     ds3231_data_t data;
00192     data.seconds = timeinfo->tm_sec;
00193     data.minutes = timeinfo->tm_min;
00194     data.hours = timeinfo->tm_hour;
00195     data.day = timeinfo->tm_wday == 0 ? 7 : timeinfo->tm_wday; // Sunday is 0 in tm struct, but 1 in
00196     // DS3231
00197     data.date = timeinfo->tm_mday;
00198     data.month = timeinfo->tm_mon + 1; // Month is 0-11 in tm struct, but 1-12 in DS3231
00199     data.year = timeinfo->tm_year - 100; // Year is since 1900, we want the last two digits
00200     data.century = timeinfo->tm_year >= 2000;
00201
00202     return set_time(&data);
00203 }
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214 time_t DS3231::get_unix_time() {
00215     ds3231_data_t data;
00216     if (get_time(&data)) {
00217         return -1;
00218     }
00219
00220     struct tm timeinfo;
00221     timeinfo.tm_sec = data.seconds;
00222     timeinfo.tm_min = data.minutes;
00223     timeinfo.tm_hour = data.hours;
00224     timeinfo.tm_mday = data.date;
00225     timeinfo.tm_mon = data.month - 1; // Month is 0-11 in tm struct, but 1-12 in DS3231
00226     timeinfo.tm_year = data.year + 100; // Year is since 1900
00227
00228     // mktime assumes that tm_wday and tm_yday are uninitialized
00229     timeinfo.tm_wday = 0;
00230     timeinfo.tm_yday = 0;
00231     timeinfo.tm_isdst = 0; // Set to 0 to use UTC
00232
00233     time_t timestamp = mktime(&timeinfo);
00234     if (timestamp == (time_t)(-1)) {
00235         uart_print("Error: mktime() failed", VerbosityLevel::ERROR);
00236         return -1;
00237     }

```

```
00238
00239     return timestamp;
00240 }
00241
00242
00251 int DS3231::clock_enable() {
00252     std::string status;
00253     uint8_t control_reg = 0;
00254     int result = i2c_read_reg(DS3231_CONTROL_REG, 1, &control_reg);
00255     if (result != 0) {
00256         status = "Failed to read control register";
00257         uart_print(status, VerbosityLevel::ERROR);
00258         return -1;
00259     }
00260
00261     // Clear the EOSC bit to enable the oscillator
00262     control_reg &= ~(1 << 7);
00263
00264     result = i2c_write_reg(DS3231_CONTROL_REG, 1, &control_reg);
00265     if (result != 0) {
00266         status = "Failed to write control register";
00267         uart_print(status, VerbosityLevel::ERROR);
00268         return -1;
00269     }
00270
00271     return 0;
00272 }
00273
00274
00283 int16_t DS3231::get_timezone_offset() const {
00284     return timezone_offset_minutes_;
00285 }
00286
00287
00298 void DS3231::set_timezone_offset(int16_t offset_minutes) {
00299     // Validate range: -12 hours to +12 hours (-720 to +720 minutes)
00300     if (offset_minutes >= -720 && offset_minutes <= 720) {
00301         timezone_offset_minutes_ = offset_minutes;
00302     } else {
00303         uart_print("Error: Invalid timezone offset", VerbosityLevel::ERROR);
00304     }
00305 }
00306
00307
00315 uint32_t DS3231::get_clock_sync_interval() const {
00316     return sync_interval_minutes_;
00317 }
00318
00319
00329 void DS3231::set_clock_sync_interval(uint32_t interval_minutes) {
00330     if (interval_minutes >= 1 && interval_minutes <= 43200) {
00331         sync_interval_minutes_ = interval_minutes;
00332     } else {
00333         uart_print("Error: Invalid sync interval", VerbosityLevel::ERROR);
00334     }
00335 }
00336
00337
00346 time_t DS3231::get_last_sync_time() const {
00347     return last_sync_time_;
00348 }
00349
00350
00359 void DS3231::update_last_sync_time() {
00360     last_sync_time_ = get_unix_time();
00361     uart_print("Clock sync time updated: " + std::to_string(last_sync_time_), VerbosityLevel::INFO);
00362 }
00363
00364
00372 time_t DS3231::get_local_time() {
00373     time_t utc_time = get_unix_time();
00374     if (utc_time == -1) {
00375         return -1;
00376     }
00377
00378     return utc_time + (timezone_offset_minutes_ * 60);
00379 }
00380
00381
00391 bool DS3231::is_sync_needed() {
00392     if (last_sync_time_ == 0) {
00393         return true;
00394     }
00395
00396     time_t current_time = get_unix_time();
00397     if (current_time == -1) {
00398         return true;
```

```

00399     }
00400
00401     time_t time_since_last_sync = current_time - last_sync_time_;
00402     uint32_t minutes_since_last_sync = time_since_last_sync / 60;
00403
00404     return minutes_since_last_sync >= sync_interval_minutes_;
00405 }
00406
00407
00411 bool DS3231::sync_clock_with_gps() {
00412     extern NMEAData nmea_data;
00413
00414     if (!nmea_data.has_valid_time()) {
00415         uart_print("GPS time data not available for sync", VerbosityLevel::WARNING);
00416         EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC_DATA_NOT_READY);
00417         return false;
00418     }
00419
00420     time_t gps_time = nmea_data.get_unix_time();
00421     if (gps_time <= 0) {
00422         uart_print("Invalid GPS time for sync", VerbosityLevel::ERROR);
00423         EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC_DATA_NOT_READY);
00424         return false;
00425     }
00426
00427     if (set_unix_time(gps_time) != 0) {
00428         uart_print("Failed to set system time from GPS", VerbosityLevel::ERROR);
00429         EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC_DATA_NOT_READY);
00430         return false;
00431     }
00432
00433     update_last_sync_time();
00434
00435     EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC);
00436     uart_print("Clock synced with GPS time: " + std::to_string(gps_time), VerbosityLevel::INFO);
00437
00438     return true;
00439 }
00440
00441 // ===== private methods
00442
00446 int DS3231::i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
00447     if (!length)
00448         return -1;
00449
00450     std::string status = "Reading register " + std::to_string(reg_addr) + " from DS3231";
00451     uart_print(status, VerbosityLevel::DEBUG);
00452     recursive_mutex_enter_blocking(&clock_mutex_);
00453     uint8_t reg = reg_addr;
00454     int write_result = i2c_write_blocking(i2c, ds3231_addr, &reg, 1, true);
00455     if (write_result == PICO_ERROR_GENERIC) {
00456         status = "Failed to write register address to DS3231";
00457         uart_print(status, VerbosityLevel::ERROR);
00458         recursive_mutex_exit(&clock_mutex_);
00459         return -1;
00460     }
00461     int read_result = i2c_read_blocking(i2c, ds3231_addr, data, length, false);
00462     if (read_result == PICO_ERROR_GENERIC) {
00463         status = "Failed to read register data from DS3231";
00464         uart_print(status, VerbosityLevel::ERROR);
00465         recursive_mutex_exit(&clock_mutex_);
00466         return -1;
00467     }
00468     recursive_mutex_exit(&clock_mutex_);
00469
00470     return 0;
00471 }
00472
00476 int DS3231::i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
00477     if (!length)
00478         return -1;
00479
00480     recursive_mutex_enter_blocking(&clock_mutex_);
00481     uint8_t message[length + 1];
00482     message[0] = reg_addr;
00483     for (int i = 0; i < length; i++) {
00484         message[i + 1] = data[i];
00485     }
00486     int write_result = i2c_write_blocking(i2c, ds3231_addr, message, (length + 1), false);
00487     if (write_result == PICO_ERROR_GENERIC) {
00488         uart_print("Error: i2c_write_blocking failed in i2c_write_reg", VerbosityLevel::ERROR);
00489         recursive_mutex_exit(&clock_mutex_);
00490         return -1;
00491     }
00492     recursive_mutex_exit(&clock_mutex_);
00493
00494     return 0;

```

```

00527 }
00528
00529     uint8_t DS3231::bin_to_bcd(const uint8_t data) {
00530         uint8_t ones_digit = (uint8_t)(data % 10);
00531         uint8_t tens_digit = (uint8_t)(data - ones_digit) / 10;
00532         return ((tens_digit < 4) + ones_digit);
00533     }
00534
00535     uint8_t DS3231::bcd_to_bin(const uint8_t bcd) {
00536         uint8_t ones_digit = (uint8_t)(bcd & 0x0F);
00537         uint8_t tens_digit = (uint8_t)(bcd >> 4);
00538         return (tens_digit * 10 + ones_digit);
00539     } // End of DS3231_RTC group

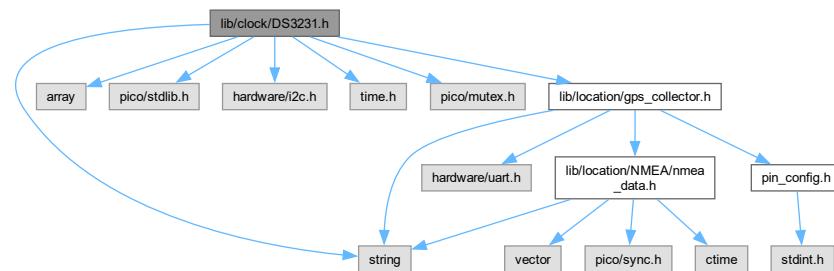
```

## 8.7 lib/clock/DS3231.h File Reference

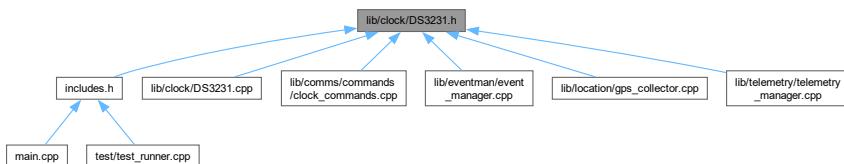
```

#include <string>
#include <array>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include <time.h>
#include "pico/mutex.h"
#include "lib/location/gps_collector.h"
Include dependency graph for DS3231.h:

```



This graph shows which files directly or indirectly include this file:



### Classes

- struct [ds3231\\_data\\_t](#)  
*Structure to hold time and date information from DS3231.*
- class [DS3231](#)  
*Class for interfacing with the DS3231 real-time clock.*

## Macros

- `#define DS3231_DEVICE_ADDRESS 0x68`  
*DS3231 I2C device address.*
- `#define DS3231_SECONDS_REG 0x00`  
*Register address: Seconds (0-59)*
- `#define DS3231_MINUTES_REG 0x01`  
*Register address: Minutes (0-59)*
- `#define DS3231_HOURS_REG 0x02`  
*Register address: Hours (0-23 in 24hr mode)*
- `#define DS3231_DAY_REG 0x03`  
*Register address: Day of the week (1-7)*
- `#define DS3231_DATE_REG 0x04`  
*Register address: Date (1-31)*
- `#define DS3231_MONTH_REG 0x05`  
*Register address: Month (1-12) & Century bit.*
- `#define DS3231_YEAR_REG 0x06`  
*Register address: Year (00-99)*
- `#define DS3231_CONTROL_REG 0x0E`  
*Register address: Control register.*
- `#define DS3231_CONTROL_STATUS_REG 0x0F`  
*Register address: Control/Status register.*
- `#define DS3231_TEMPERATURE_MSB_REG 0x11`  
*Register address: Temperature register (MSB)*
- `#define DS3231_TEMPERATURE_LSB_REG 0x12`  
*Register address: Temperature register (LSB)*

## Enumerations

- enum `days_of_week` {  
  `MONDAY = 1, TUESDAY, WEDNESDAY, THURSDAY,`  
  `FRIDAY, SATURDAY, SUNDAY` }  
*Enumeration of days of the week.*

### 8.7.1 Macro Definition Documentation

#### 8.7.1.1 DS3231\_DEVICE\_ADDRESS

```
#define DS3231_DEVICE_ADDRESS 0x68
```

`DS3231` I2C device address.

Definition at line 15 of file `DS3231.h`.

#### 8.7.1.2 DS3231\_SECONDS\_REG

```
#define DS3231_SECONDS_REG 0x00
```

Register address: Seconds (0-59)

Definition at line 20 of file `DS3231.h`.

### 8.7.1.3 DS3231\_MINUTES\_REG

```
#define DS3231_MINUTES_REG 0x01
```

Register address: Minutes (0-59)

Definition at line [25](#) of file [DS3231.h](#).

### 8.7.1.4 DS3231\_HOURS\_REG

```
#define DS3231_HOURS_REG 0x02
```

Register address: Hours (0-23 in 24hr mode)

Definition at line [30](#) of file [DS3231.h](#).

### 8.7.1.5 DS3231\_DAY\_REG

```
#define DS3231_DAY_REG 0x03
```

Register address: Day of the week (1-7)

Definition at line [35](#) of file [DS3231.h](#).

### 8.7.1.6 DS3231\_DATE\_REG

```
#define DS3231_DATE_REG 0x04
```

Register address: Date (1-31)

Definition at line [40](#) of file [DS3231.h](#).

### 8.7.1.7 DS3231\_MONTH\_REG

```
#define DS3231_MONTH_REG 0x05
```

Register address: Month (1-12) & Century bit.

Definition at line [45](#) of file [DS3231.h](#).

### 8.7.1.8 DS3231\_YEAR\_REG

```
#define DS3231_YEAR_REG 0x06
```

Register address: Year (00-99)

Definition at line [50](#) of file [DS3231.h](#).

### 8.7.1.9 DS3231\_CONTROL\_REG

```
#define DS3231_CONTROL_REG 0x0E
```

Register address: Control register.

Definition at line 55 of file [DS3231.h](#).

### 8.7.1.10 DS3231\_CONTROL\_STATUS\_REG

```
#define DS3231_CONTROL_STATUS_REG 0x0F
```

Register address: Control/Status register.

Definition at line 60 of file [DS3231.h](#).

### 8.7.1.11 DS3231\_TEMPERATURE\_MSB\_REG

```
#define DS3231_TEMPERATURE_MSB_REG 0x11
```

Register address: Temperature register (MSB)

Definition at line 65 of file [DS3231.h](#).

### 8.7.1.12 DS3231\_TEMPERATURE\_LSB\_REG

```
#define DS3231_TEMPERATURE_LSB_REG 0x12
```

Register address: Temperature register (LSB)

Definition at line 70 of file [DS3231.h](#).

## 8.7.2 Enumeration Type Documentation

### 8.7.2.1 days\_of\_week

```
enum days_of_week
```

Enumeration of days of the week.

Enumerator

MONDAY	Monday.
TUESDAY	Tuesday.
WEDNESDAY	Wednesday.
THURSDAY	Thursday.
FRIDAY	Friday.
SATURDAY	Saturday.
SUNDAY	Sunday.

Definition at line 76 of file [DS3231.h](#).

## 8.8 DS3231.h

[Go to the documentation of this file.](#)

```

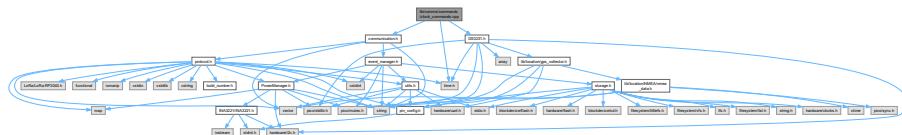
00001 #ifndef DS3231_H
00002 #define DS3231_H
00003
00004 #include <string>
00005 #include <array>
00006 #include "pico/stdlib.h"
00007 #include "hardware/i2c.h"
00008 #include <time.h>
00009 #include "pico/mutex.h"
00010 #include "lib/location/gps_collector.h"
00011
00015 #define DS3231_DEVICE_ADDRESS 0x68
00016
00020 #define DS3231_SECONDS_REG 0x00
00021
00025 #define DS3231_MINUTES_REG 0x01
00026
00030 #define DS3231_HOURS_REG 0x02
00031
00035 #define DS3231_DAY_REG 0x03
00036
00040 #define DS3231_DATE_REG 0x04
00041
00045 #define DS3231_MONTH_REG 0x05
00046
00050 #define DS3231_YEAR_REG 0x06
00051
00055 #define DS3231_CONTROL_REG 0x0E
00056
00060 #define DS3231_CONTROL_STATUS_REG 0x0F
00061
00065 #define DS3231_TEMPERATURE_MSB_REG 0x11
00066
00070 #define DS3231_TEMPERATURE_LSB_REG 0x12
00071
00076 enum days_of_week {
00077     MONDAY = 1,
00078     TUESDAY,
00079     WEDNESDAY,
00080     THURSDAY,
00081     FRIDAY,
00082     SATURDAY,
00083     SUNDAY
00084 };
00085
00090 typedef struct {
00091     uint8_t seconds;
00092     uint8_t minutes;
00093     uint8_t hours;
00094     uint8_t day;
00095     uint8_t date;
00096     uint8_t month;
00097     uint8_t year;
00098     bool century;
00099 } ds3231_data_t;
00100
00108 class DS3231 {
00109 public:
00115     DS3231(i2c_inst_t *i2c_instance);
00116
00123     int set_time(ds3231_data_t *data);
00124
00131     int get_time(ds3231_data_t *data);
00132
00139     int read_temperature(float *resolution);
00140
00147     int set_unix_time(time_t unix_time);
00148
00154     time_t get_unix_time();
00155
00161     int clock_enable();
00162
00168     int16_t get_timezone_offset() const;
00169
00175     void set_timezone_offset(int16_t offset_minutes);
00176
00182     uint32_t get_clock_sync_interval() const;
00183
00189     void set_clock_sync_interval(uint32_t interval_minutes);
00190
00196     time_t get_last_sync_time() const;
00197

```

```
00201     void update_last_sync_time();
00202
00208     time_t get_local_time();
00209
00215     bool is_sync_needed();
00216
00222     bool sync_clock_with_gps();
00223
00224
00225 private:
00226     i2c_inst_t *i2c;
00227     uint8_t ds3231_addr;
00228
00237     int i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data);
00238
00247     int i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data);
00248
00255     uint8_t bin_to_bcd(const uint8_t data);
00256
00263     uint8_t bcd_to_bin(const uint8_t bcd);
00264
00265     recursive_mutex_t clock_mutex_;
00266     int16_t timezone_offset_minutes_ = 60;
00267     uint32_t sync_interval_minutes_ = 1440;
00268     time_t last_sync_time_ = 0;
00269 };
00270
00271 #endif // DS3231_H
```

## 8.9 lib/comms/commands/clock\_commands.cpp File Reference

```
#include "communication.h"
#include <time.h>
#include "DS3231.h"
Include dependency graph for clock_commands.cpp:
```



## Macros

- #define CLOCK\_GROUP 3
  - #define TIME 0
  - #define TIMEZONE\_OFFSET 1
  - #define CLOCK\_SYNC\_INTERVAL 2
  - #define LAST\_SYNC\_TIME 3

## Functions

- std::vector< Frame > handle\_time (const std::string &param, OperationType operationType)  
*Handler for getting and setting system time.*
  - std::vector< Frame > handle\_timezone\_offset (const std::string &param, OperationType operationType)  
*Handler for getting and setting timezone offset.*
  - std::vector< Frame > handle\_clock\_sync\_interval (const std::string &param, OperationType operationType)  
*Handler for getting and setting clock synchronization interval.*
  - std::vector< Frame > handle\_get\_last\_sync\_time (const std::string &param, OperationType operationType)  
*Handler for getting last clock sync time.*

**Variables**

- DS3231 systemClock

**8.9.1 Macro Definition Documentation****8.9.1.1 CLOCK\_GROUP**

```
#define CLOCK_GROUP 3
```

Definition at line 5 of file [clock\\_commands.cpp](#).

**8.9.1.2 TIME**

```
#define TIME 0
```

Definition at line 6 of file [clock\\_commands.cpp](#).

**8.9.1.3 TIMEZONE\_OFFSET**

```
#define TIMEZONE_OFFSET 1
```

Definition at line 7 of file [clock\\_commands.cpp](#).

**8.9.1.4 CLOCK\_SYNC\_INTERVAL**

```
#define CLOCK_SYNC_INTERVAL 2
```

Definition at line 8 of file [clock\\_commands.cpp](#).

**8.9.1.5 LAST\_SYNC\_TIME**

```
#define LAST_SYNC_TIME 3
```

Definition at line 9 of file [clock\\_commands.cpp](#).

## 8.10 clock\_commands.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002 #include <time.h>
00003 #include "DS3231.h" // Include the DS3231 header
00004
00005 #define CLOCK_GROUP 3
00006 #define TIME 0
00007 #define TIMEZONE_OFFSET 1
00008 #define CLOCK_SYNC_INTERVAL 2
00009 #define LAST_SYNC_TIME 3
0010
0016
0017 extern DS3231 systemClock;
0018
0032 std::vector<Frame> handle_time(const std::string& param, OperationType operationType) {
0033     std::vector<Frame> frames;
0034     std::string error_msg;
0035
0036     if (operationType == OperationType::SET) {
0037         if (param.empty()) {
0038             error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
0039             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
0040             return frames;
0041         }
0042         try {
0043             time_t newTime = std::stoll(param);
0044             if (newTime <= 0) {
0045                 error_msg = error_code_to_string(ErrorCode::INVALID_VALUE);
0046                 frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
0047                 return frames;
0048             }
0049
0050             if (systemClock.set_unix_time(newTime) != 0) {
0051                 error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET);
0052                 frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
0053                 return frames;
0054             }
0055
0056             EventEmitter::emit(EventGroup::CLOCK, ClockEvent::CHANGED);
0057             frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, TIME,
0058                 std::to_string(systemClock.get_unix_time())));
0059             return frames;
0060         } catch (...) {
0061             error_msg = error_code_to_string(ErrorCode::INVALID_FORMAT);
0062             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
0063             return frames;
0064         }
0064     } else if (operationType == OperationType::GET) {
0065         if (!param.empty()) {
0066             error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
0067             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
0068             return frames;
0069         }
0070
0071         uint32_t time_unix = systemClock.get_local_time();
0072         if (time_unix == 0) {
0073             error_msg = error_code_to_string(ErrorCode::INTERNAL_FAIL_TO_READ);
0074             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
0075             return frames;
0076         }
0077
0078         frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, TIME,
0079             std::to_string(time_unix)));
0080         return frames;
0081
0082     error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
0083     frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
0084     return frames;
0085 }
0086
0097 std::vector<Frame> handle_timezone_offset(const std::string& param, OperationType operationType) {
0098     std::vector<Frame> frames;
0099     std::string error_msg;
0100
0101     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
0102         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
0103         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
0104         return frames;
0105     }
0106
0107     if (operationType == OperationType::GET) {
0108         if (!param.empty()) {

```

```

00109         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00110         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET,
00111             error_msg));
00112     }
00113
00114     extern DS3231 systemclock;
00115     int offset = systemClock.get_timezone_offset();
00116     std::string offset_set = std::to_string(offset);
00117     frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, TIMEZONE_OFFSET, offset_set));
00118     return frames;
00119 }
00120
00121 if (param.empty()) {
00122     error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
00123     frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
00124     return frames;
00125 }
00126
00127 try {
00128     int16_t offset = std::stoi(param);
00129     if (offset < -720 || offset > 720) {
00130         error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
00131         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET,
00132             error_msg));
00133         return frames;
00134     }
00135
00136     extern DS3231 systemClock;
00137     systemClock.set_timezone_offset(offset);
00138     std::string offset_set = std::to_string(offset);
00139     frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, TIMEZONE_OFFSET, offset_set));
00140     return frames;
00141 } catch (...) {
00142     error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
00143     frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
00144 }
00145 }
00146
00147
00148 std::vector<Frame> handle_clock_sync_interval(const std::string& param, OperationType operationType) {
00149     std::vector<Frame> frames;
00150     std::string error_msg;
00151
00152     if (!operationType == OperationType::GET || operationType == OperationType::SET)) {
00153         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00154         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00155             error_msg));
00156         return frames;
00157     }
00158
00159     if (operationType == OperationType::GET) {
00160         if (!param.empty()) {
00161             error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00162             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00163                 error_msg));
00164             return frames;
00165         }
00166
00167         extern DS3231 systemClock;
00168         uint32_t syncInterval = systemClock.get_clock_sync_interval();
00169         std::string clockSyncInterval = std::to_string(syncInterval);
00170         frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00171             clockSyncInterval));
00172         return frames;
00173     }
00174
00175     if (operationType == OperationType::SET) {
00176         if (param.empty()) {
00177             error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
00178             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00179                 error_msg));
00180             return frames;
00181         }
00182         try {
00183             uint32_t interval = std::stoul(param);
00184
00185             extern DS3231 systemClock;
00186             systemClock.set_clock_sync_interval(interval);
00187             std::string interval_set = std::to_string(interval);
00188
00189             frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00190                 interval_set));
00191             return frames;
00192         } catch (...) {
00193             error_msg = error_code_to_string(ErrorCode::INVALID_VALUE);
00194         }
00195     }
00196 }
```

```

00199         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00200             error_msg));
00201     }
00202     error_msg = error_code_to_string(ErrorCode::UNKNOWN_ERROR);
00203     frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, error_msg));
00204     return frames;
00205 }
00206
00216 std::vector<Frame> handle_get_last_sync_time(const std::string& param, OperationType operationType) {
00217     std::vector<Frame> frames;
00218     std::string error_msg;
00219
00220     if (operationType != OperationType::GET || !param.empty()) {
00221         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00222         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, LAST_SYNC_TIME, error_msg));
00223     }
00224
00225     extern DS3231 systemClock;
00226     time_t lastSyncTime = systemClock.get_last_sync_time();
00227
00228     if (lastSyncTime == 0) {
00229         frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, LAST_SYNC_TIME, "NEVER"));
00230     } else {
00231         frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, LAST_SYNC_TIME,
00232             std::to_string(lastSyncTime)));
00233     }
00234
00235
00236     return frames;
00237 } // end of ClockCommands group

```

## 8.11 lib/comms/commands/commands.cpp File Reference

```
#include "commands.h"
#include "communication.h"
Include dependency graph for commands.cpp:
```



### Typedefs

- using **CommandHandler** = std::function<std::vector<Frame>(const std::string&, OperationType)>  
*Function type for command handlers.*
- using **CommandMap** = std::map<uint32\_t, CommandHandler>  
*Map type for storing command handlers.*

### Functions

- std::vector< Frame > execute\_command (uint32\_t commandKey, const std::string &param, OperationType operationType)  
*Executes a command based on its key.*

### Variables

- **CommandMap command\_handlers**  
*Global map of all command handlers.*

## 8.12 commands.cpp

[Go to the documentation of this file.](#)

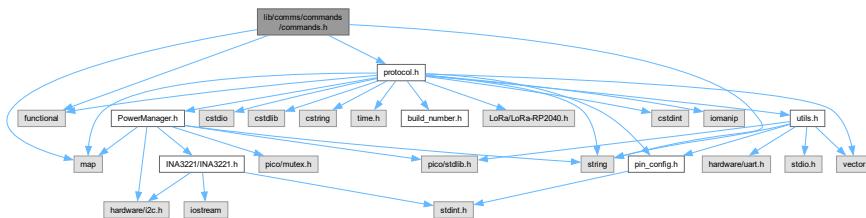
```

00001 // commands/commands.cpp
00002 #include "commands.h"
00003 #include "communication.h"
00004
00010
00015 using CommandHandler = std::function<std::vector<Frame>(&const std::string, OperationType)>;
00016
00021 using CommandMap = std::map<uint32_t, CommandHandler>;
00022
00027 CommandMap command_handlers = {
00028     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(0)), handle_get_commands_list},
00029     // Group 1, Command 0
00030     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(1)), handle_get_build_version},
00031     // Group 1, Command 1
00032     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(8)), handle_verbosity},
00033     // Group 1, Command 9
00034     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(9)), handle_enter_bootloader_mode},
00035     // Group 1, Command 9
00036     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(0)), handle_get_power_manager_ids},
00037     // Group 2, Command 0
00038     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(2)), handle_get_voltage_battery},
00039     // Group 2, Command 2
00040     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(3)), handle_get_voltage_5v},
00041     // Group 2, Command 3
00042     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(4)), handle_get_current_charge_usb},
00043     // Group 2, Command 4
00044     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(5)), handle_get_current_charge_solar},
00045     // Group 2, Command 5
00046     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(6)), handle_get_current_charge_total},
00047     // Group 2, Command 6
00048     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(7)), handle_get_current_draw},
00049     // Group 2, Command 7
00050     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(0)), handle_time},
00051     // Group 3, Command 0
00052     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(1)), handle_timezone_offset},
00053     // Group 3, Command 1
00054     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(2)), handle_clock_sync_interval},
00055     // Group 3, Command 2
00056     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(3)), handle_get_last_sync_time},
00057     // Group 3, Command 3
00058     {((static_cast<uint32_t>(4) << 8) | static_cast<uint32_t>(0)), handle_get_sensor_data},
00059     // Group 4, Command 0
00060     {((static_cast<uint32_t>(4) << 8) | static_cast<uint32_t>(1)), handle_sensor_config},
00061     // Group 4, Command 1
00062     {((static_cast<uint32_t>(4) << 8) | static_cast<uint32_t>(3)), handle_get_sensor_list},
00063     // Group 4, Command 3
00064     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(1)), handle_get_last_events},
00065     // Group 5, Command 1
00066     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(2)), handle_get_event_count},
00067     // Group 5, Command 2
00068     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(0)), handle_list_files},
00069     // Group 6, Command 0
00070     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(4)), handle_mount},
00071     // Group 6, Command 4
00072     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(1)), handle_gps_power_status},
00073     // Group 7, Command 1
00074     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(2)), handle_enable_gps_uart_passthrough},
00075     // Group 7, Command 3
00076     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(3)), handle_get_rmc_data},
00077     // Group 7, Command 4
00078     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(4)), handle_get_gga_data},
00079 };
00080
00081 std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType
00082 operationType) {
00083     auto it = command_handlers.find(commandKey);
00084     if (it != command_handlers.end()) {
00085         CommandHandler handler = it->second;
00086         return handler(param, operationType);
00087     } else {
00088         std::vector<Frame> frames;
00089         frames.push_back(frame_build(OperationType::ERR, 0, 0, "INVALID COMMAND"));
00090         return frames;
00091     }
00092 } // end of CommandSystem group

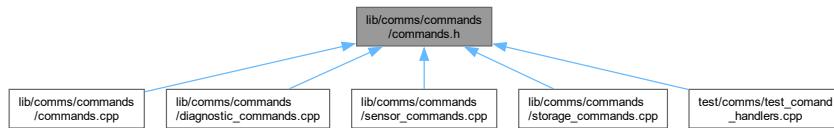
```

## 8.13 lib/comms/commands/commands.h File Reference

```
#include <string>
#include <functional>
#include <map>
#include "protocol.h"
Include dependency graph for commands.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- `std::vector< Frame > handle_time (const std::string &param, OperationType operationType)`  
*Handler for getting and setting system time.*
- `std::vector< Frame > handle_timezone_offset (const std::string &param, OperationType operationType)`  
*Handler for getting and setting timezone offset.*
- `std::vector< Frame > handle_clock_sync_interval (const std::string &param, OperationType operationType)`  
*Handler for getting and setting clock synchronization interval.*
- `std::vector< Frame > handle_get_last_sync_time (const std::string &param, OperationType operationType)`  
*Handler for getting last clock sync time.*
- `std::vector< Frame > handle_get_commands_list (const std::string &param, OperationType operationType)`  
*Handler for listing all available commands on UART.*
- `std::vector< Frame > handle_get_build_version (const std::string &param, OperationType operationType)`  
*Get firmware build version.*
- `std::vector< Frame > handle_verbosity (const std::string &param, OperationType operationType)`  
*Handles setting or getting the UART verbosity level.*
- `std::vector< Frame > handle_enter_bootloader_mode (const std::string &param, OperationType operationType)`  
*Reboot system to USB firmware loader.*
- `std::vector< Frame > handle_gps_power_status (const std::string &param, OperationType operationType)`  
*Handler for controlling GPS module power state.*
- `std::vector< Frame > handle_enable_gps_uart_passthrough (const std::string &param, OperationType operationType)`

- std::vector< Frame > **handle\_get\_rmc\_data** (const std::string &param, OperationType operationType)
 

*Handler for enabling GPS transparent mode (UART pass-through)*
- std::vector< Frame > **handle\_get\_gga\_data** (const std::string &param, OperationType operationType)
 

*Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.*
- std::vector< Frame > **handle\_get\_power\_manager\_ids** (const std::string &param, OperationType operationType)
 

*Handler for retrieving Power Manager IDs.*
- std::vector< Frame > **handle\_get\_voltage\_battery** (const std::string &param, OperationType operationType)
 

*Handler for getting battery voltage.*
- std::vector< Frame > **handle\_get\_voltage\_5v** (const std::string &param, OperationType operationType)
 

*Handler for getting 5V rail voltage.*
- std::vector< Frame > **handle\_get\_current\_charge\_usb** (const std::string &param, OperationType operationType)
 

*Handler for getting USB charge current.*
- std::vector< Frame > **handle\_get\_current\_charge\_solar** (const std::string &param, OperationType operationType)
 

*Handler for getting solar panel charge current.*
- std::vector< Frame > **handle\_get\_current\_charge\_total** (const std::string &param, OperationType operationType)
 

*Handler for getting total charge current.*
- std::vector< Frame > **handle\_get\_current\_draw** (const std::string &param, OperationType operationType)
 

*Handler for getting system current draw.*
- std::vector< Frame > **handle\_get\_last\_events** (const std::string &param, OperationType operationType)
 

*Handler for retrieving last N events from the event log.*
- std::vector< Frame > **handle\_get\_event\_count** (const std::string &param, OperationType operationType)
 

*Handler for getting total number of events in the log.*
- std::vector< Frame > **handle\_list\_files** (const std::string &param, OperationType operationType)
 

*Handles the list files command.*
- std::vector< Frame > **handle\_mount** (const std::string &param, OperationType operationType)
 

*Handles the SD card mount/unmount command.*
- std::vector< Frame > **handle\_get\_sensor\_data** (const std::string &param, OperationType operationType)
 

*Handler for reading sensor data.*
- std::vector< Frame > **handle\_sensor\_config** (const std::string &param, OperationType operationType)
 

*Handler for configuring sensors.*
- std::vector< Frame > **handle\_get\_sensor\_list** (const std::string &param, OperationType operationType)
 

*Handler for listing available sensors.*
- std::vector< Frame > **execute\_command** (uint32\_t commandKey, const std::string &param, OperationType operationType)
 

*Executes a command based on its key.*

## Variables

- std::map< uint32\_t, std::function< std::vector< Frame > (const std::string &, OperationType)> > **command\_handlers**

*Global map of all command handlers.*

## 8.14 commands.h

[Go to the documentation of this file.](#)

```

00001 // commands/commands.h
00002 #ifndef COMMANDS_H
00003 #define COMMANDS_H
00004
00005 #include <string>
00006 #include <functional>
00007 #include <map>
00008 #include "protocol.h"
00009
00010 // CLOCK
00011 std::vector<Frame> handle_time(const std::string& param, OperationType operationType);
00012 std::vector<Frame> handle_timezone_offset(const std::string& param, OperationType operationType);
00013 std::vector<Frame> handle_clock_sync_interval(const std::string& param, OperationType operationType);
00014 std::vector<Frame> handle_get_last_sync_time(const std::string& param, OperationType operationType);
00015
00016
00017 // DIAG
00018 std::vector<Frame> handle_get_commands_list(const std::string& param, OperationType operationType);
00019 std::vector<Frame> handle_get_build_version(const std::string& param, OperationType operationType);
00020 std::vector<Frame> handle_verbosity(const std::string& param, OperationType operationType);
00021 std::vector<Frame> handle_enter_bootloader_mode(const std::string& param, OperationType
operationType);
00022
00023
00024 // GPS
00025 std::vector<Frame> handle_gps_power_status(const std::string& param, OperationType operationType);
00026 std::vector<Frame> handle_enable_gps_uart_passthrough(const std::string& param, OperationType
operationType);
00027 std::vector<Frame> handle_get_rmc_data(const std::string& param, OperationType operationType);
00028 std::vector<Frame> handle_get_gga_data(const std::string& param, OperationType operationType);
00029
00030
00031 // POWER
00032 std::vector<Frame> handle_get_power_manager_ids(const std::string& param, OperationType
operationType);
00033 std::vector<Frame> handle_get_voltage_battery(const std::string& param, OperationType operationType);
00034 std::vector<Frame> handle_get_voltage_5v(const std::string& param, OperationType operationType);
00035 std::vector<Frame> handle_get_current_charge_usb(const std::string& param, OperationType
operationType);
00036 std::vector<Frame> handle_get_current_charge_solar(const std::string& param, OperationType
operationType);
00037 std::vector<Frame> handle_get_current_charge_total(const std::string& param, OperationType
operationType);
00038 std::vector<Frame> handle_get_current_draw(const std::string& param, OperationType operationType);
00039
00040
00041 // EVENT
00042 std::vector<Frame> handle_get_last_events(const std::string& param, OperationType operationType);
00043 std::vector<Frame> handle_get_event_count(const std::string& param, OperationType operationType);
00044
00045
00046 // STORAGE
00047 std::vector<Frame> handle_list_files(const std::string& param, OperationType operationType);
00048 std::vector<Frame> handle_mount(const std::string& param, OperationType operationType);
00049
00050 // SENSOR
00051 std::vector<Frame> handle_get_sensor_data(const std::string& param, OperationType operationType);
00052 std::vector<Frame> handle_sensor_config(const std::string& param, OperationType operationType);
00053 std::vector<Frame> handle_get_sensor_list(const std::string& param, OperationType operationType);
00054
00055 std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType
operationType);
00056 extern std::map<uint32_t, std::function<std::vector<Frame>(const std::string&, OperationType)>>
command_handlers;
00057
00058 #endif

```

## 8.15 lib/comms/commands/diagnostic\_commands.cpp File Reference

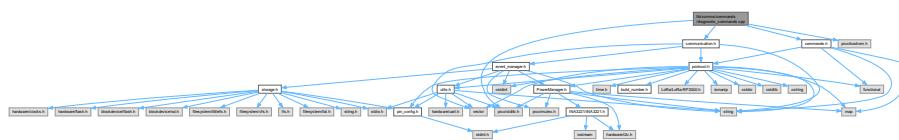
```

#include "communication.h"
#include "commands.h"
#include "pico/stdc.h"

```

```
#include "pico/bootrom.h"
```

Include dependency graph for diagnostic\_commands.cpp:



## Functions

- std::vector< Frame > **handle\_get\_commands\_list** (const std::string &param, OperationType operationType)  
*Handler for listing all available commands on UART.*
- std::vector< Frame > **handle\_get\_build\_version** (const std::string &param, OperationType operationType)  
*Get firmware build version.*
- std::vector< Frame > **handle\_set\_verbosity** (const std::string &param, OperationType operationType)  
*Handles setting or getting the UART verbosity level.*
- std::vector< Frame > **handle\_enter\_bootloader\_mode** (const std::string &param, OperationType operationType)  
*Reboot system to USB firmware loader.*

## Variables

- volatile bool **g\_pending\_bootloader\_reset**

## 8.16 diagnostic\_commands.cpp

Go to the documentation of this file.

```
00001 #include "communication.h"
00002 #include "commands.h"
00003 #include "pico/stl.h"
00004 #include "pico/bootrom.h"
00005
00010
00011 extern volatile bool g_pending_bootloader_reset;
00012
00023 std::vector<Frame> handle_get_commands_list(const std::string& param, OperationType operationType) {
00024     std::vector<Frame> frames;
00025     std::string error_msg;
00026
00027     if (!param.empty()) {
00028         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00029         frames.push_back(frame_build(OperationType::ERR, 1, 0, error_msg));
00030         return frames;
00031     }
00032
00033     if (!(operationType == OperationType::GET)) {
00034         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00035         frames.push_back(frame_build(OperationType::ERR, 1, 0, error_msg));
00036         return frames;
00037     }
00038
00039     std::string combined_command_details;
00040     for (const auto& entry : command_handlers) {
00041         uint32_t command_key = entry.first;
00042         uint8_t group = (command_key >> 8) & 0xFF;
00043         uint8_t command = command_key & 0xFF;
00044
00045         std::string command_details = std::to_string(group) + "." + std::to_string(command);
00046
00047         if (combined_command_details.length() + command_details.length() + 1 > 100) {
00048             frames.push_back(frame_build(OperationType::SEQ, 1, 0, combined_command_details));
```

```
00049         combined_command_details = "";
00050     }
00051
00052     if (!combined_command_details.empty()) {
00053         combined_command_details += "-";
00054     }
00055     combined_command_details += command_details;
00056 }
00057
00058 if (!combined_command_details.empty()) {
00059     frames.push_back(frame_build(OperationType::SEQ, 1, 0, combined_command_details));
00060 }
00061
00062 frames.push_back(frame_build(OperationType::VAL, 1, 0, "SEQ_DONE"));
00063
00064 }
00065
00066
00077 std::vector<Frame> handle_get_build_version(const std::string& param, OperationType operationType) {
00078     std::vector<Frame> frames;
00079     std::string error_msg;
00080
00081     if (!param.empty()) {
00082         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00083         frames.push_back(frame_build(OperationType::ERR, 1, 1, error_msg));
00084         return frames;
00085     }
00086
00087     if (operationType == OperationType::GET) {
00088         frames.push_back(frame_build(OperationType::VAL, 1, 1, std::to_string(BUILD_NUMBER)));
00089         return frames;
00090     }
00091
00092     error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00093     frames.push_back(frame_build(OperationType::ERR, 1, 1, error_msg));
00094
00095 }
00096
00097
00119 std::vector<Frame> handle_verbosity(const std::string& param, OperationType operationType) {
00120     std::vector<Frame> frames;
00121     std::string error_msg;
00122
00123     if (operationType == OperationType::GET && param.empty()) {
00124         uart_print("Current verbosity level: " + std::to_string(static_cast<int>(g_uart_verbosity)),
00125                    VerbosityLevel::INFO);
00126         frames.push_back(frame_build(OperationType::VAL, 1, 8,
00127                                     std::to_string(static_cast<int>(g_uart_verbosity))));
00128         return frames;
00129     }
00130
00131     try {
00132         int level = std::stoi(param);
00133         if (level < 0 || level > 5) {
00134             error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
00135             frames.push_back(frame_build(OperationType::ERR, 1, 8, error_msg));
00136             return frames;
00137         }
00138         g_uart_verbosity = static_cast<VerbosityLevel>(level);
00139         frames.push_back(frame_build(OperationType::RES, 1, 8, "SET " + std::to_string(level)));
00140         return frames;
00141     } catch (...) {
00142         error_msg = error_code_to_string(ErrorCode::INVALID_FORMAT);
00143         frames.push_back(frame_build(OperationType::ERR, 1, 8, error_msg));
00144         return frames;
00145     }
00146 }
00147
00158 std::vector<Frame> handle_enter_bootloader_mode(const std::string& param, OperationType operationType)
{
00159     std::vector<Frame> frames;
00160     std::string error_msg;
00161
00162     if (param != "USB") {
00163         error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
00164         frames.push_back(frame_build(OperationType::ERR, 1, 9, error_msg));
00165         return frames;
00166     }
00167
00168     if (operationType != OperationType::SET) {
00169         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00170         frames.push_back(frame_build(OperationType::ERR, 1, 9, error_msg));
00171         return frames;
00172     }
00173
00174     frames.push_back(frame_build(OperationType::RES, 1, 9, "REBOOT BOOTSEL"));
00175 }
```

```

00176     // Set flag to trigger reboot after function returns
00177     g_pending_bootloader_reset = true;
00178
00179     return frames;
00180 }
00181

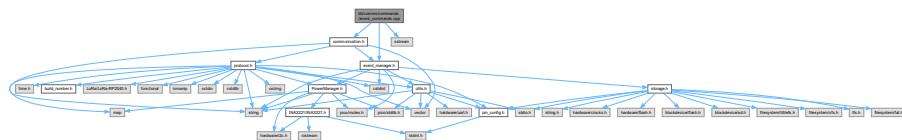
```

## 8.17 lib/comms/commands/event\_commands.cpp File Reference

```

#include "communication.h"
#include "event_manager.h"
#include <sstream>
Include dependency graph for event_commands.cpp:

```



### Functions

- std::vector< Frame > handle\_get\_last\_events (const std::string &param, OperationType operationType)  
*Handler for retrieving last N events from the event log.*
- std::vector< Frame > handle\_get\_event\_count (const std::string &param, OperationType operationType)  
*Handler for getting total number of events in the log.*

## 8.18 event\_commands.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002 #include "event_manager.h"
00003 #include <sstream>
00004
00005
00011
00033 std::vector<Frame> handle_get_last_events(const std::string& param, OperationType operationType) {
00034     std::vector<Frame> frames;
00035     std::string error_msg;
00036
00037     if (operationType != OperationType::GET) {
00038         error_msg = error_code_to_string(KeyCode::INVALID_OPERATION);
00039         frames.push_back(frame_build(OperationType::ERR, 5, 1, error_msg));
00040         return frames;
00041     }
00042     size_t count = 10; // Default number of events to return
00043     if (!param.empty()) {
00044         try {
00045             count = std::stoul(param);
00046             if (count > EVENT_BUFFER_SIZE) {
00047                 error_msg = error_code_to_string(KeyCode::INVALID_VALUE);
00048                 frames.push_back(frame_build(OperationType::ERR, 5, 1, error_msg));
00049                 return frames;
00050             }
00051         } catch (...) {
00052             error_msg = error_code_to_string(KeyCode::PARAM_INVALID);
00053             frames.push_back(frame_build(OperationType::ERR, 5, 1, error_msg));
00054             return frames;
00055         }
00056     }
00057     size_t available = eventManager.get_event_count();

```

```
00059     size_t to_return = (count == 0) ? available : std::min(count, available);
00060     size_t event_index = available;
00061
00062     while (to_return > 0) {
00063         std::stringstream ss;
00064         ss << std::hex << std::uppercase << std::setfill('0');
00065         size_t events_in_frame = 0;
00066
00067         for (size_t i = 0; i < 10 && to_return > 0; ++i) {
00068             event_index--;
00069             const EventLog& event = eventManager.get_event(event_index);
00070
00071             ss << std::setw(4) << event.id
00072                 << std::setw(8) << event.timestamp
00073                 << std::setw(2) << static_cast<int>(event.group)
00074                 << std::setw(2) << static_cast<int>(event.event);
00075
00076             if (to_return > 1) ss << "-";
00077
00078             to_return--;
00079             events_in_frame++;
00080         }
00081         frames.push_back(frame_build(OperationType::SEQ, 5, 1, ss.str()));
00082     }
00083     frames.push_back(frame_build(OperationType::VAL, 5, 1, "SEQ_DONE"));
00084     return frames;
00085 }
00086
00087
00100    std::vector<Frame> handle_get_event_count(const std::string& param, OperationType operationType) {
00101        std::vector<Frame> frames;
00102        std::string error_msg;
00103
00104        if (operationType != OperationType::GET || !param.empty()) {
00105            error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00106            frames.push_back(frame_build(OperationType::ERR, 5, 2, error_msg));
00107            return frames;
00108        }
00109        frames.push_back(frame_build(OperationType::VAL, 5, 2,
00110            std::to_string(eventManager.get_event_count())));
00111    } // end of EventCommands group
```

## 8.19 lib/comms/commands/gps\_commands.cpp File Reference

```
#include "communication.h"
#include "lib/location/gps_collector.h"
#include <sstream>
Include dependency graph for gps_commands.cpp:
```



## Macros

- #define GPS\_GROUP 7
  - #define POWER\_STATUS\_COMMAND 1
  - #define PASSTHROUGH\_COMMAND 2
  - #define RMC\_DATA\_COMMAND 3
  - #define GGA\_DATA\_COMMAND 4

## Functions

- std::vector< Frame > handle\_gps\_power\_status (const std::string &param, OperationType operationType)  
*Handler for controlling GPS module power state.*
- std::vector< Frame > handle\_enable\_gps\_uart\_passthrough (const std::string &param, OperationType operationType)  
*Handler for enabling GPS transparent mode (UART pass-through)*
- std::vector< Frame > handle\_get\_rmc\_data (const std::string &param, OperationType operationType)  
*Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.*
- std::vector< Frame > handle\_get\_gga\_data (const std::string &param, OperationType operationType)  
*Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.*

### 8.19.1 Macro Definition Documentation

#### 8.19.1.1 GPS\_GROUP

```
#define GPS_GROUP 7
```

Definition at line 5 of file [gps\\_commands.cpp](#).

#### 8.19.1.2 POWER\_STATUS\_COMMAND

```
#define POWER_STATUS_COMMAND 1
```

Definition at line 6 of file [gps\\_commands.cpp](#).

#### 8.19.1.3 PASSTHROUGH\_COMMAND

```
#define PASSTHROUGH_COMMAND 2
```

Definition at line 7 of file [gps\\_commands.cpp](#).

#### 8.19.1.4 RMC\_DATA\_COMMAND

```
#define RMC_DATA_COMMAND 3
```

Definition at line 8 of file [gps\\_commands.cpp](#).

#### 8.19.1.5 GGA\_DATA\_COMMAND

```
#define GGA_DATA_COMMAND 4
```

Definition at line 9 of file [gps\\_commands.cpp](#).

## 8.20 gps\_commands.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002 #include "lib/location/gps_collector.h"
00003 #include <iostream> // Include for stringstream
00004
00005 #define GPS_GROUP 7
00006 #define POWER_STATUS_COMMAND 1
00007 #define PASSTHROUGH_COMMAND 2
00008 #define RMC_DATA_COMMAND 3
00009 #define GGA_DATA_COMMAND 4
0010
0016
0032 std::vector<Frame> handle_gps_power_status(const std::string& param, OperationType operationType) {
0033     std::vector<Frame> frames;
0034     std::string error_str;
0035
0036     if (operationType == OperationType::SET) {
0037         if (param.empty()) {
0038             error_str = error_code_to_string(ErrorCode::PARAM_REQUIRED);
0039             frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND,
0040                                 error_str));
0040             return frames;
0041         }
0042
0043         try {
0044             int power_status = std::stoi(param);
0045             if (power_status != 0 && power_status != 1) {
0046                 error_str = error_code_to_string(ErrorCode::PARAM_INVALID);
0047                 frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND,
0048                                     error_str));
0048                 return frames;
0049             }
0050             gpio_put(GPS_POWER_ENABLE_PIN, power_status);
0051             EventEmitter::emit(EventGroup::GPS, power_status ? GPSEvent::POWER_ON :
0051                               GPSEvent::POWER_OFF);
0052             frames.push_back(frame_build(OperationType::RES, GPS_GROUP, POWER_STATUS_COMMAND,
0053                                 std::to_string(power_status)));
0053             return frames;
0054         } catch (...) {
0055             error_str = error_code_to_string(ErrorCode::PARAM_INVALID);
0056             frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND,
0057                                 error_str));
0057             return frames;
0058         }
0059     }
0060
0061     // GET operation
0062     if (!param.empty()) {
0063         error_str = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
0064         frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND, error_str));
0065         return frames;
0066     }
0067
0068     bool power_status = gpio_get(GPS_POWER_ENABLE_PIN);
0069     frames.push_back(frame_build(OperationType::VAL, GPS_GROUP, POWER_STATUS_COMMAND,
0070                         std::to_string(power_status)));
0070     return frames;
0071 }
0072
0073
0089 std::vector<Frame> handle_enable_gps_uart_passthrough(const std::string& param, OperationType
0090 operationType) {
0090     std::vector<Frame> frames;
0091     std::string error_str;
0092
0093     if (!(operationType == OperationType::SET)) {
0094         error_str = error_code_to_string(ErrorCode::INVALID_OPERATION);
0095         frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, PASSTHROUGH_COMMAND, error_str));
0096         return frames;
0097     }
0098
0099     // Parse and validate timeout parameter
0100     uint32_t timeout_ms;
0101     try {
0102         timeout_ms = param.empty() ? 60000u : std::stoul(param) * 1000;
0103     } catch (...) {
0104         error_str = error_code_to_string(ErrorCode::INVALID_VALUE);
0105         frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, PASSTHROUGH_COMMAND, error_str));
0106         return frames;
0107     }
0108
0109     // Setup UART parameters and exit sequence
0110     const std::string EXIT_SEQUENCE = "##EXIT##";

```

```

00111     std::string input_buffer;
00112     bool exit_requested = false;
00113     static volatile bool pause_gps_collection = false;
00114     pause_gps_collection = true;
00115     sleep_ms(100);
00116
00117     uint32_t original_baud_rate = DEBUG_UART_BAUD_RATE;
00118     uint32_t gps_baud_rate = GPS_UART_BAUD_RATE;
00119     uint32_t start_time = to_ms_since_boot(get_absolute_time());
00120
00121     EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_START);
00122
00123     std::string message = "Entering GPS Serial Pass-Through Mode @" +
00124         std::to_string(gps_baud_rate) + " for " +
00125         std::to_string(timeout_ms/1000) + "s\r\n" +
00126         "Send " + EXIT_SEQUENCE + " to exit";
00127     uart_print(message, VerbosityLevel::INFO);
00128
00129     sleep_ms(10);
00130
00131     uart_set_baudrate(DEBUG_UART_PORT, gps_baud_rate);
00132
00133     while (!exit_requested) {
00134         while (uart_is_readable(DEBUG_UART_PORT)) {
00135             char ch = uart_getc(DEBUG_UART_PORT);
00136
00137             input_buffer += ch;
00138             if (input_buffer.length() > EXIT_SEQUENCE.length()) {
00139                 input_buffer = input_buffer.substr(1);
00140             }
00141
00142             if (input_buffer == EXIT_SEQUENCE) {
00143                 exit_requested = true;
00144                 break;
00145             }
00146
00147             if (input_buffer != EXIT_SEQUENCE.substr(0, input_buffer.length())) {
00148                 uart_write_blocking(GPS_UART_PORT,
00149                     reinterpret_cast<const uint8_t*>(&ch), 1);
00150             }
00151         }
00152
00153         while (uart_is_readable(GPS_UART_PORT)) {
00154             char gps_byte = uart_getc(GPS_UART_PORT);
00155             uart_write_blocking(DEBUG_UART_PORT,
00156                 reinterpret_cast<const uint8_t*>(&gps_byte), 1);
00157         }
00158
00159         if (to_ms_since_boot(get_absolute_time()) - start_time >= timeout_ms) {
00160             break;
00161         }
00162     }
00163
00164     uart_set_baudrate(DEBUG_UART_PORT, original_baud_rate);
00165
00166     sleep_ms(50);
00167
00168     pause_gps_collection = false;
00169
00170     EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_END);
00171
00172     std::string exit_reason = exit_requested ? "USER_EXIT" : "TIMEOUT";
00173     std::string response = "GPS UART BRIDGE EXIT: " + exit_reason;
00174     uart_print(response, VerbosityLevel::INFO);
00175
00176     frames.push_back(frame_build(OperationType::RES, GPS_GROUP, PASSTHROUGH_COMMAND, response));
00177     return frames;
00178 }
00179
00180
00193 std::vector<Frame> handle_get_rmc_data(const std::string& param, OperationType operationType) {
00194     std::vector<Frame> frames;
00195     std::string error_msg;
00196
00197     if (operationType != OperationType::GET) {
00198         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00199         frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, RMC_DATA_COMMAND, error_msg));
00200         return frames;
00201     }
00202
00203     if (!param.empty()) {
00204         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00205         frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, RMC_DATA_COMMAND, error_msg));
00206         return frames;
00207     }
00208
00209     std::vector<std::string> tokens = nmea_data.get_rmc_tokens();

```

```

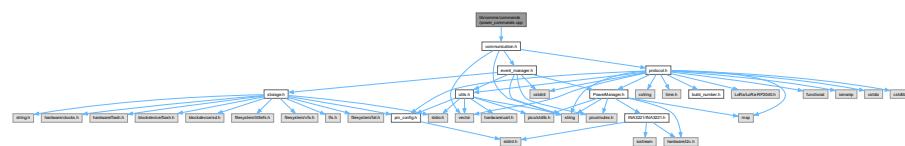
00210     std::stringstream ss;
00211     for (size_t i = 0; i < tokens.size(); ++i) {
00212         ss « tokens[i];
00213         if (i < tokens.size() - 1) {
00214             ss « ",";
00215         }
00216     }
00217
00218     frames.push_back(frame_build(OperationType::VAL, GPS_GROUP, RMC_DATA_COMMAND, ss.str()));
00219     return frames;
00220 }
00221
00222
00223 std::vector<Frame> handle_get_gga_data(const std::string& param, OperationType operationType) {
00224     std::vector<Frame> frames;
00225     std::string error_msg;
00226
00227     if (operationType != OperationType::GET) {
00228         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00229         frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, GGA_DATA_COMMAND, error_msg));
00230         return frames;
00231     }
00232
00233     if (!param.empty()) {
00234         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00235         frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, GGA_DATA_COMMAND, error_msg));
00236         return frames;
00237     }
00238
00239     std::vector<std::string> tokens = nmea_data.get_gga_tokens();
00240     std::stringstream ss;
00241     for (size_t i = 0; i < tokens.size(); ++i) {
00242         ss « tokens[i];
00243         if (i < tokens.size() - 1) {
00244             ss « ",";
00245         }
00246     }
00247
00248     frames.push_back(frame_build(OperationType::VAL, GPS_GROUP, GGA_DATA_COMMAND, ss.str()));
00249     return frames;
00250 }
00251
00252 // end of GPSCommands group

```

## 8.21 lib/comms/commands/power\_commands.cpp File Reference

#include "communication.h"

Include dependency graph for power\_commands.cpp:



### Macros

- #define POWER\_GROUP 2
- #define POWER\_MANAGER\_IDS 0
- #define VOLTAGE\_BATTERY 2
- #define VOLTAGE\_MAIN 3
- #define CHARGE\_USB 4
- #define CHARGE\_SOLAR 5
- #define CHARGE\_TOTAL 6
- #define DRAW\_TOTAL 7

## Functions

- std::vector< Frame > handle\_get\_power\_manager\_ids (const std::string &param, OperationType operationType)
   
*Handler for retrieving Power Manager IDs.*
- std::vector< Frame > handle\_get\_voltage\_battery (const std::string &param, OperationType operationType)
   
*Handler for getting battery voltage.*
- std::vector< Frame > handle\_get\_voltage\_5v (const std::string &param, OperationType operationType)
   
*Handler for getting 5V rail voltage.*
- std::vector< Frame > handle\_get\_current\_charge\_usb (const std::string &param, OperationType operationType)
   
*Handler for getting USB charge current.*
- std::vector< Frame > handle\_get\_current\_charge\_solar (const std::string &param, OperationType operationType)
   
*Handler for getting solar panel charge current.*
- std::vector< Frame > handle\_get\_current\_charge\_total (const std::string &param, OperationType operationType)
   
*Handler for getting total charge current.*
- std::vector< Frame > handle\_get\_current\_draw (const std::string &param, OperationType operationType)
   
*Handler for getting system current draw.*

### 8.21.1 Macro Definition Documentation

#### 8.21.1.1 POWER\_GROUP

```
#define POWER_GROUP 2
```

Definition at line 3 of file [power\\_commands.cpp](#).

#### 8.21.1.2 POWER\_MANAGER\_IDS

```
#define POWER_MANAGER_IDS 0
```

Definition at line 4 of file [power\\_commands.cpp](#).

#### 8.21.1.3 VOLTAGE\_BATTERY

```
#define VOLTAGE_BATTERY 2
```

Definition at line 5 of file [power\\_commands.cpp](#).

#### 8.21.1.4 VOLTAGE\_MAIN

```
#define VOLTAGE_MAIN 3
```

Definition at line 6 of file [power\\_commands.cpp](#).

### 8.21.1.5 CHARGE\_USB

```
#define CHARGE_USB 4
```

Definition at line 7 of file [power\\_commands.cpp](#).

### 8.21.1.6 CHARGE\_SOLAR

```
#define CHARGE_SOLAR 5
```

Definition at line 8 of file [power\\_commands.cpp](#).

### 8.21.1.7 CHARGE\_TOTAL

```
#define CHARGE_TOTAL 6
```

Definition at line 9 of file [power\\_commands.cpp](#).

### 8.21.1.8 DRAW\_TOTAL

```
#define DRAW_TOTAL 7
```

Definition at line 10 of file [power\\_commands.cpp](#).

## 8.22 power\_commands.cpp

[Go to the documentation of this file.](#)

```
00001 #include "communication.h"
00002
00003 #define POWER_GROUP 2
00004 #define POWER_MANAGER_IDS 0
00005 #define VOLTAGE_BATTERY 2
00006 #define VOLTAGE_MAIN 3
00007 #define CHARGE_USB 4
00008 #define CHARGE_SOLAR 5
00009 #define CHARGE_TOTAL 6
00010 #define DRAW_TOTAL 7
00011
00012
00030 std::vector<Frame> handle_get_power_manager_ids(const std::string& param, OperationType operationType)
{
00031     std::vector<Frame> frames;
00032     std::string error_msg;
00033
00034     if (!param.empty()) {
00035         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00036         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, POWER_MANAGER_IDS, error_msg));
00037         return frames;
00038     }
00039
00040     if (!(operationType == OperationType::GET)) {
00041         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00042         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, POWER_MANAGER_IDS, error_msg));
00043         return frames;
00044     }
00045
00046     extern PowerManager powerManager;
00047     std::string power_manager_ids = powerManager.read_device_ids();
00048     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, POWER_MANAGER_IDS,
00049     power_manager_ids));
00049     return frames;
```

```

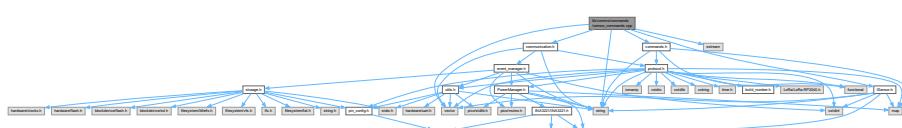
00050 }
00051
00064 std::vector<Frame> handle_get_voltage_battery(const std::string& param, OperationType operationType) {
00065     std::vector<Frame> frames;
00066     std::string error_msg;
00067
00068     if (!param.empty()) {
00069         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00070         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_BATTERY, error_msg));
00071         return frames;
00072     }
00073
00074     if (!(operationType == OperationType::GET)) {
00075         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00076         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_BATTERY, error_msg));
00077         return frames;
00078     }
00079
00080     extern PowerManager powerManager;
00081     float voltage = powerManager.get_voltage_battery();
00082     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, VOLTAGE_BATTERY,
00083         std::to_string(voltage), ValueUnit::VOLT));
00084     return frames;
00085
00098 std::vector<Frame> handle_get_voltage_5v(const std::string& param, OperationType operationType) {
00099     std::vector<Frame> frames;
00100     std::string error_msg;
00101
00102     if (!param.empty()) {
00103         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00104         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_MAIN, error_msg));
00105         return frames;
00106     }
00107
00108     if (!(operationType == OperationType::GET)) {
00109         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00110         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_MAIN, error_msg));
00111         return frames;
00112     }
00113
00114     extern PowerManager powerManager;
00115     float voltage = powerManager.get_voltage_5v();
00116     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, VOLTAGE_MAIN,
00117         std::to_string(voltage), ValueUnit::VOLT));
00118     return frames;
00119
00132 std::vector<Frame> handle_get_current_charge_usb(const std::string& param, OperationType
00133     operationType) {
00134     std::vector<Frame> frames;
00135     std::string error_msg;
00136
00137     if (!param.empty()) {
00138         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00139         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_USB, error_msg));
00140         return frames;
00141
00142     if (!(operationType == OperationType::GET)) {
00143         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00144         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_USB, error_msg));
00145         return frames;
00146     }
00147
00148     extern PowerManager powerManager;
00149     float charge_current = powerManager.get_current_charge_usb();
00150     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, CHARGE_USB,
00151         std::to_string(charge_current), ValueUnit::MILIAMP));
00152     return frames;
00153
00166 std::vector<Frame> handle_get_current_charge_solar(const std::string& param, OperationType
00167     operationType) {
00168     std::vector<Frame> frames;
00169     std::string error_msg;
00170
00171     if (!param.empty()) {
00172         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00173         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_SOLAR, error_msg));
00174         return frames;
00175
00176     if (!(operationType == OperationType::GET)) {
00177         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00178         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_SOLAR, error_msg));
00179         return frames;

```

```
00180 }
00181
00182     extern PowerManager powerManager;
00183     float charge_current = powerManager.get_current_charge_solar();
00184     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, CHARGE_SOLAR,
00185         std::to_string(charge_current), ValueUnit::MILIAMP));
00186 }
00187
00188 std::vector<Frame> handle_get_current_charge_total(const std::string& param, OperationType
00189 operationType) {
00200     std::vector<Frame> frames;
00201     std::string error_msg;
00202
00203     if (!param.empty()) {
00204         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00205         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_TOTAL, error_msg));
00206         return frames;
00207     }
00208 }
00209
00210     if (!(operationType == OperationType::GET)) {
00211         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00212         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_TOTAL, error_msg));
00213         return frames;
00214     }
00215
00216     extern PowerManager powerManager;
00217     float charge_current = powerManager.get_current_charge_total();
00218     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, CHARGE_TOTAL,
00219         std::to_string(charge_current), ValueUnit::MILIAMP));
00220 }
00221
00222 std::vector<Frame> handle_get_current_draw(const std::string& param, OperationType operationType) {
00223     std::vector<Frame> frames;
00224     std::string error_msg;
00225
00226     if (!param.empty()) {
00227         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00228         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, DRAW_TOTAL, error_msg));
00229         return frames;
00230     }
00231
00232     if (!(operationType == OperationType::GET)) {
00233         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00234         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, DRAW_TOTAL, error_msg));
00235         return frames;
00236     }
00237
00238     extern PowerManager powerManager;
00239     float current_draw = powerManager.get_current_draw();
00240     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, DRAW_TOTAL,
00241         std::to_string(current_draw), ValueUnit::MILIAMP));
00242 }
00243
00244 // end of PowerCommands group
```

## 8.23 lib/comms/commands/sensor commands.cpp File Reference

```
#include "communication.h"
#include "ISensor.h"
#include <vector>
#include <string>
#include <iostream>
#include "commands.h"
```



## Macros

- #define SENSOR\_GROUP 4
- #define SENSOR\_READ 0
- #define SENSOR\_CONFIGURE 1

## Functions

- std::vector< Frame > handle\_get\_sensor\_data (const std::string &param, OperationType operationType)  
*Handler for reading sensor data.*
- std::vector< Frame > handle\_sensor\_config (const std::string &param, OperationType operationType)  
*Handler for configuring sensors.*
- std::vector< Frame > handle\_get\_sensor\_list (const std::string &param, OperationType operationType)  
*Handler for listing available sensors.*

### 8.23.1 Macro Definition Documentation

#### 8.23.1.1 SENSOR\_GROUP

```
#define SENSOR_GROUP 4
```

Definition at line 8 of file [sensor\\_commands.cpp](#).

#### 8.23.1.2 SENSOR\_READ

```
#define SENSOR_READ 0
```

Definition at line 9 of file [sensor\\_commands.cpp](#).

#### 8.23.1.3 SENSOR\_CONFIGURE

```
#define SENSOR_CONFIGURE 1
```

Definition at line 10 of file [sensor\\_commands.cpp](#).

## 8.24 sensor\_commands.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002 #include "ISensor.h"
00003 #include <vector>
00004 #include <string>
00005 #include <iostream>
00006 #include "commands.h"
00007
00008 #define SENSOR_GROUP 4
00009 #define SENSOR_READ 0
0010 #define SENSOR_CONFIGURE 1
0011
0017
0036 std::vector<Frame> handle_get_sensor_data(const std::string& param, OperationType operationType) {
0037     std::vector<Frame> frames;
0038     std::string error_msg;
0039
0040     if (param.empty()) {
0041         error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
0042         frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
0043         return frames;
0044     }
0045
0046     if (operationType != OperationType::GET) {
0047         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
0048         frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
0049         return frames;
0050     }
0051
0052     // Parse sensor type and data type from param
0053     std::string sensor_type_str;
0054     std::string data_type_str;
0055
0056     size_t dash_pos = param.find('-');
0057     if (dash_pos != std::string::npos) {
0058         sensor_type_str = param.substr(0, dash_pos);
0059         data_type_str = param.substr(dash_pos + 1);
0060     } else {
0061         sensor_type_str = param;
0062     }
0063
0064     // Convert sensor_type_str to SensorType
0065     SensorType sensor_type;
0066     if (sensor_type_str == "light") {
0067         sensor_type = SensorType::LIGHT;
0068     } else if (sensor_type_str == "environment") {
0069         sensor_type = SensorType::ENVIRONMENT;
0070     } else if (sensor_type_str == "magnetometer") {
0071         sensor_type = SensorType::MAGNETOMETER;
0072     } else if (sensor_type_str == "imu") {
0073         sensor_type = SensorType::IMU;
0074     } else {
0075         error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Invalid sensor type";
0076         frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
0077         return frames;
0078     }
0079
0080     SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();
0081
0082     // If data type is specified, read that specific data
0083     if (!data_type_str.empty()) {
0084         SensorDataTypeIdentifier data_type;
0085
0086         // Map string to SensorDataTypeIdentifier
0087         if (data_type_str == "light_level") {
0088             data_type = SensorDataTypeIdentifier::LIGHT_LEVEL;
0089         } else if (data_type_str == "temperature") {
0090             data_type = SensorDataTypeIdentifier::TEMPERATURE;
0091         } else if (data_type_str == "pressure") {
0092             data_type = SensorDataTypeIdentifier::PRESSURE;
0093         } else if (data_type_str == "humidity") {
0094             data_type = SensorDataTypeIdentifier::HUMIDITY;
0095         } else if (data_type_str == "mag_field_x") {
0096             data_type = SensorDataTypeIdentifier::MAG_FIELD_X;
0097         } else if (data_type_str == "mag_field_y") {
0098             data_type = SensorDataTypeIdentifier::MAG_FIELD_Y;
0099         } else if (data_type_str == "mag_field_z") {
0100             data_type = SensorDataTypeIdentifier::MAG_FIELD_Z;
0101         } else if (data_type_str == "gyro_x") {
0102             data_type = SensorDataTypeIdentifier::GYRO_X;
0103         } else if (data_type_str == "gyro_y") {
0104             data_type = SensorDataTypeIdentifier::GYRO_Y;
0105         } else if (data_type_str == "gyro_z") {
0106

```

```

00106     data_type = SensorDataTypeIdentifier::GYRO_Z;
00107 } else if (data_type_str == "accel_x") {
00108     data_type = SensorDataTypeIdentifier::ACCEL_X;
00109 } else if (data_type_str == "accel_y") {
00110     data_type = SensorDataTypeIdentifier::ACCEL_Y;
00111 } else if (data_type_str == "accel_z") {
00112     data_type = SensorDataTypeIdentifier::ACCEL_Z;
00113 } else {
00114     error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Invalid data type";
00115     frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
00116     return frames;
00117 }
00118
00119     float value = sensor_wrapper.sensor_read_data(sensor_type, data_type);
00120     std::stringstream ss;
00121     ss << value;
00122     frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, SENSOR_READ, ss.str()));
00123 }
00124 // If only sensor type is specified, read all relevant data for that sensor type
00125 else {
00126     std::vector<SensorDataTypeIdentifier> data_types;
00127
00128     switch (sensor_type) {
00129         case SensorType::LIGHT:
00130             data_types = {SensorDataTypeIdentifier::LIGHT_LEVEL};
00131             break;
00132         case SensorType::ENVIRONMENT:
00133             data_types = {
00134                 SensorDataTypeIdentifier::TEMPERATURE,
00135                 SensorDataTypeIdentifier::PRESSURE,
00136                 SensorDataTypeIdentifier::HUMIDITY
00137             };
00138             break;
00139         case SensorType::MAGNETOMETER:
00140             data_types = {
00141                 SensorDataTypeIdentifier::MAG_FIELD_X,
00142                 SensorDataTypeIdentifier::MAG_FIELD_Y,
00143                 SensorDataTypeIdentifier::MAG_FIELD_Z
00144             };
00145             break;
00146         case SensorType::IMU:
00147             data_types = {
00148                 SensorDataTypeIdentifier::GYRO_X,
00149                 SensorDataTypeIdentifier::GYRO_Y,
00150                 SensorDataTypeIdentifier::GYRO_Z,
00151                 SensorDataTypeIdentifier::ACCEL_X,
00152                 SensorDataTypeIdentifier::ACCEL_Y,
00153                 SensorDataTypeIdentifier::ACCEL_Z
00154             };
00155             break;
00156     }
00157
00158     std::stringstream combined_values;
00159     std::vector<std::string> data_type_names;
00160     std::vector<float> values;
00161
00162 // Get names for the data types and store the values
00163 for (SensorDataTypeIdentifier data_type : data_types) {
00164     switch (data_type) {
00165         case SensorDataTypeIdentifier::LIGHT_LEVEL:
00166             data_type_names.push_back("light_level");
00167             break;
00168         case SensorDataTypeIdentifier::TEMPERATURE:
00169             data_type_names.push_back("temperature");
00170             break;
00171         case SensorDataTypeIdentifier::PRESSURE:
00172             data_type_names.push_back("pressure");
00173             break;
00174         case SensorDataTypeIdentifier::HUMIDITY:
00175             data_type_names.push_back("humidity");
00176             break;
00177         case SensorDataTypeIdentifier::MAG_FIELD_X:
00178             data_type_names.push_back("mag_field_x");
00179             break;
00180         case SensorDataTypeIdentifier::MAG_FIELD_Y:
00181             data_type_names.push_back("mag_field_y");
00182             break;
00183         case SensorDataTypeIdentifier::MAG_FIELD_Z:
00184             data_type_names.push_back("mag_field_z");
00185             break;
00186         case SensorDataTypeIdentifier::GYRO_X:
00187             data_type_names.push_back("gyro_x");
00188             break;
00189         case SensorDataTypeIdentifier::GYRO_Y:
00190             data_type_names.push_back("gyro_y");
00191             break;
00192         case SensorDataTypeIdentifier::GYRO_Z:
00193             data_type_names.push_back("gyro_z");
00194     }
00195 }
```

```

00193             data_type_names.push_back("gyro_z");
00194             break;
00195         case SensorDataTypeIdentifier::ACCEL_X:
00196             data_type_names.push_back("accel_x");
00197             break;
00198         case SensorDataTypeIdentifier::ACCEL_Y:
00199             data_type_names.push_back("accel_y");
00200             break;
00201         case SensorDataTypeIdentifier::ACCEL_Z:
00202             data_type_names.push_back("accel_z");
00203             break;
00204     }
00205
00206     float value = sensor_wrapper.sensor_read_data(sensor_type, data_type);
00207     values.push_back(value);
00208 }
00209
00210 // Format output as key-value pairs
00211 for (size_t i = 0; i < data_type_names.size(); i++) {
00212     if (i > 0) combined_values « "|";
00213     combined_values « data_type_names[i] « ":" « values[i];
00214 }
00215
00216 frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, SENSOR_READ,
combined_values.str()));
00217 }
00218
00219 return frames;
00220 }
00221
00222 std::vector<Frame> handle_sensor_config(const std::string& param, OperationType operationType) {
00223     std::vector<Frame> frames;
00224     std::string error_msg;
00225
00226     if (param.empty()) {
00227         error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
00228         frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
00229         return frames;
00230     }
00231
00232     if (operationType != OperationType::SET) {
00233         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00234         frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
00235         return frames;
00236     }
00237
00238     // Parse sensor type and configuration from param
00239     size_t semicolon_pos = param.find(';');
00240     if (semicolon_pos == std::string::npos) {
00241         error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Format should be
sensor_type;config_params";
00242         frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
00243         return frames;
00244     }
00245
00246     std::string sensor_type_str = param.substr(0, semicolon_pos);
00247     std::string config_str = param.substr(semicolon_pos + 1);
00248
00249     // Convert sensor_type_str to SensorType
00250     SensorType sensor_type;
00251     if (sensor_type_str == "light") {
00252         sensor_type = SensorType::LIGHT;
00253     } else if (sensor_type_str == "environment") {
00254         sensor_type = SensorType::ENVIRONMENT;
00255     } else if (sensor_type_str == "magnetometer") {
00256         sensor_type = SensorType::MAGNETOMETER;
00257     } else if (sensor_type_str == "imu") {
00258         sensor_type = SensorType::IMU;
00259     } else {
00260         error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Invalid sensor type";
00261         frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
00262         return frames;
00263     }
00264
00265     // Parse configuration parameters
00266     std::map<std::string, std::string> config_map;
00267     std::stringstream ss(config_str);
00268     std::string config_pair;
00269
00270     while (std::getline(ss, config_pair, '|')) {
00271         size_t colon_pos = config_pair.find(':');
00272         if (colon_pos != std::string::npos) {
00273             std::string key = config_pair.substr(0, colon_pos);
00274             std::string value = config_pair.substr(colon_pos + 1);
00275             config_map[key] = value;
00276         }
00277     }
00278 }
```

```

00292
00293 // Apply configuration
00294 SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();
00295 bool success = sensor_wrapper.sensor_configure(sensor_type, config_map);
00296
00297 if (success) {
00298     frames.push_back(frame_build(OperationType::RES, SENSOR_GROUP, SENSOR_CONFIGURE,
00299     "Configuration successful"));
00300 } else {
00301     error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET) + ": Failed to configure sensor";
00302     frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
00303 }
00304 return frames;
00305 }
00306
00307
00320 std::vector<Frame> handle_get_sensor_list(const std::string& param, OperationType operationType) {
00321     std::vector<Frame> frames;
00322     std::string error_msg;
00323
00324     if (operationType != OperationType::GET)
00325         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00326     frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, 2, error_msg));
00327     return frames;
00328 }
00329
00330 // Get the singleton instance
00331 SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();
00332
00333 // Get list of available sensor types
00334 std::vector<std::pair<SensorType, uint8_t>> available_sensors =
00335     sensor_wrapper.get_available_sensors();
00336
00337 if (available_sensors.empty()) {
00338     frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, 2, "No sensors available"));
00339     return frames;
00340 }
00341 std::stringstream sensor_list;
00342 bool first = true;
00343
00344 for (const auto& sensor_info : available_sensors) {
00345     if (!first) {
00346         sensor_list << "|";
00347     }
00348
00349     // Format: sensor_type:address (in hex)
00350     std::stringstream addr_hex;
00351     addr_hex << std::hex << static_cast<int>(sensor_info.second);
00352
00353     switch (sensor_info.first) {
00354         case SensorType::LIGHT:
00355             sensor_list << "light:0x" << addr_hex.str();
00356             break;
00357         case SensorType::ENVIRONMENT:
00358             sensor_list << "environment:0x" << addr_hex.str();
00359             break;
00360         case SensorType::MAGNETOMETER:
00361             sensor_list << "magnetometer:0x" << addr_hex.str();
00362             break;
00363         case SensorType::IMU:
00364             sensor_list << "imu:0x" << addr_hex.str();
00365             break;
00366         default:
00367             sensor_list << "unknown:0x" << addr_hex.str();
00368             break;
00369     }
00370
00371     first = false;
00372 }
00373
00374 frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, 2, sensor_list.str()));
00375 return frames;
00376 }

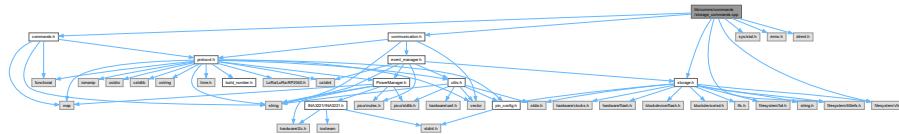
```

## 8.25 lib/comms/commands/storage\_commands.cpp File Reference

```
#include "commands.h"
#include "communication.h"
```

```
#include "storage.h"
#include "filesystem/vfs.h"
#include "filesystem/littlefs.h"
#include <sys/stat.h>
#include <errno.h>
#include "dirent.h"

Include dependency graph for storage_commands.cpp:
```



## Macros

- #define STORAGE\_GROUP 6
- #define LIST\_FILES\_COMMAND 0
- #define MOUNT\_COMMAND 4

## Functions

- std::vector< Frame > handle\_list\_files (const std::string &param, OperationType operationType)  
*Handles the list files command.*
- std::vector< Frame > handle\_mount (const std::string &param, OperationType operationType)  
*Handles the SD card mount/unmount command.*

### 8.25.1 Macro Definition Documentation

#### 8.25.1.1 STORAGE\_GROUP

```
#define STORAGE_GROUP 6
```

Definition at line 10 of file [storage\\_commands.cpp](#).

#### 8.25.1.2 LIST\_FILES\_COMMAND

```
#define LIST_FILES_COMMAND 0
```

Definition at line 12 of file [storage\\_commands.cpp](#).

#### 8.25.1.3 MOUNT\_COMMAND

```
#define MOUNT_COMMAND 4
```

Definition at line 13 of file [storage\\_commands.cpp](#).

## 8.26 storage\_commands.cpp

[Go to the documentation of this file.](#)

```

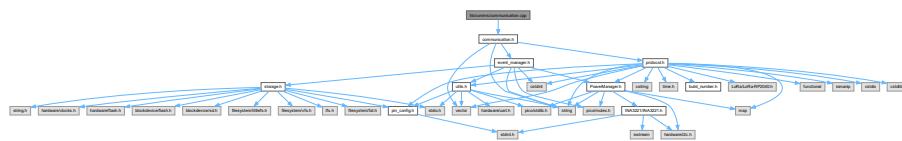
00001 #include "commands.h"
00002 #include "communication.h"
00003 #include "storage.h"
00004 #include "filesystem/vfs.h"
00005 #include "filesystem/littlefs.h"
00006 #include <sys/stat.h>
00007 #include <errno.h>
00008 #include "dirent.h"
00009
00010 #define STORAGE_GROUP 6
00011
00012 #define LIST_FILES_COMMAND 0
00013 #define MOUNT_COMMAND 4
00019
00037 std::vector<Frame> handle_list_files(const std::string& param, OperationType operationType) {
00038     std::vector<Frame> frames;
00039     std::string error_msg;
00040
00041     if (operationType != OperationType::GET) {
00042         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00043         frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND,
00044             error_msg));
00045         return frames;
00046     }
00047
00048     DIR* dir;
00049     struct dirent* ent;
00050     int file_count = 0; // Counter for the number of files
00051     if ((dir = opendir(".")) != NULL) {
00052         // First, count the number of files
00053         while ((ent = readdir(dir)) != NULL) {
00054             const char* filename = ent->d_name;
00055             if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0) {
00056                 continue;
00057             }
00058             file_count++;
00059         }
00060         closedir(dir);
00061
00062         // Send the number of files
00063         frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, LIST_FILES_COMMAND,
00064             std::to_string(file_count)));
00065
00066         // Open the directory again to read file information
00067         dir = opendir("/");
00068         if (dir != NULL) {
00069             while ((ent = readdir(dir)) != NULL) {
00070                 const char* filename = ent->d_name;
00071
00072                 // Skip "." and ".." directories
00073                 if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0) {
00074                     continue;
00075
00076                 // Get file size
00077                 char filepath[256];
00078                 snprintf(filepath, sizeof(filepath), "/%s", filename);
00079
00080                 FILE* file = fopen(filepath, "rb");
00081                 size_t file_size = 0;
00082
00083                 if (file != NULL) {
00084                     fseek(file, 0, SEEK_END);
00085                     file_size = ftell(file);
00086                     fclose(file);
00087                 }
00088
00089                 // Create and send frame with filename and size
00090                 char file_info[512];
00091                 snprintf(file_info, sizeof(file_info), "%s:%zu", filename, file_size);
00092                 frames.push_back(frame_build(OperationType::SEQ, STORAGE_GROUP, LIST_FILES_COMMAND,
00093                     file_info));
00094
00095                 closedir(dir);
00096                 frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, LIST_FILES_COMMAND,
00097                     "SEQ_DONE"));
00098
00099             }
00100         }
00101     }
00102     return frames;
00103 } else {
00104     error_msg = error_code_to_string(ErrorCode::INTERNAL_FAIL_TO_READ);
00105     frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND,
00106         error_msg));
00107 }
00108
00109 return frames;

```

```
00100     }
00101 } else {
00102     frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND,
00103     error_msg));
00104     return frames;
00105 }
00106
00123 std::vector<Frame> handle_mount(const std::string& param, OperationType operationType) {
00124     std::vector<Frame> frames;
00125     std::string error_msg;
00126
00127     if (operationType == OperationType::GET) {
00128         frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, MOUNT_COMMAND,
00129         std::to_string(sd_card_mounted)));
00130         return frames;
00131     } else if (operationType == OperationType::SET) {
00132         if (param == "1") {
00133             if (fs_init()) {
00134                 frames.push_back(frame_build(OperationType::RES, STORAGE_GROUP, MOUNT_COMMAND,
00135                 "SD_MOUNT_OK"));
00136                 return frames;
00137             } else {
00138                 error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET);
00139                 frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND,
00140                 error_msg));
00141                 return frames;
00142             }
00143         } else if (param == "0") {
00144             if (fs_unmount("/") == 0) {
00145                 sd_card_mounted = false;
00146                 frames.push_back(frame_build(OperationType::RES, STORAGE_GROUP, MOUNT_COMMAND,
00147                 "SD_UNMOUNT_OK"));
00148                 return frames;
00149             } else {
00150                 error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET);
00151                 frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND,
00152                 error_msg));
00153                 return frames;
00154             }
00155         } else {
00156             error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
00157             frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND,
00158             error_msg));
00159         }
00160 } // StorageCommands
```

## 8.27 lib/comms/communication.cpp File Reference

```
#include "communication.h"
Include dependency graph for communication.cpp:
```



## Functions

- bool `initialize_radio()`

*Initializes the LoRa radio module.*

## Variables

- string `outgoing`
- uint8\_t `msgCount` = 0
- long `lastSendTime` = 0
- long `lastReceiveTime` = 0
- long `lastPrintTime` = 0
- unsigned long `interval` = 0

## 8.27.1 Function Documentation

### 8.27.1.1 initialize\_radio()

```
bool initialize_radio ()
```

Initializes the LoRa radio module.

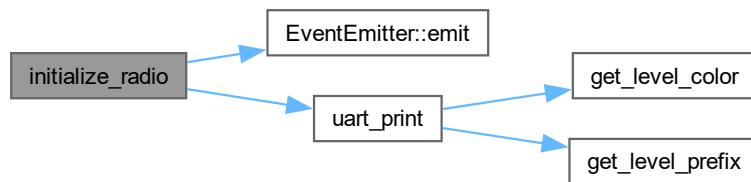
#### Returns

True if initialization was successful, false otherwise.

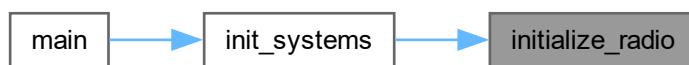
Sets the LoRa pins and attempts to begin LoRa communication at a specified frequency. Emits a `CommsEvent::RADIO_INIT` event on success or a `CommsEvent::RADIO_ERROR` event on failure.

Definition at line 17 of file `communication.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



## 8.27.2 Variable Documentation

### 8.27.2.1 outgoing

```
string outgoing
```

Definition at line [3](#) of file [communication.cpp](#).

### 8.27.2.2 msgCount

```
uint8_t msgCount = 0
```

Definition at line [4](#) of file [communication.cpp](#).

### 8.27.2.3 lastSendTime

```
long lastSendTime = 0
```

Definition at line [5](#) of file [communication.cpp](#).

### 8.27.2.4 lastReceiveTime

```
long lastReceiveTime = 0
```

Definition at line [6](#) of file [communication.cpp](#).

### 8.27.2.5 lastPrintTime

```
long lastPrintTime = 0
```

Definition at line [7](#) of file [communication.cpp](#).

### 8.27.2.6 interval

```
unsigned long interval = 0
```

Definition at line [8](#) of file [communication.cpp](#).

## 8.28 communication.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002
00003 string outgoing;
00004 uint8_t msgCount = 0;
00005 long lastSendTime = 0;
00006 long lastReceiveTime = 0;
00007 long lastPrintTime = 0;
00008 unsigned long interval = 0;
00009
00010
00011 bool initialize_radio() {
00012     LoRa.set_pins(lora_cs_pin, lora_reset_pin, lora_irq_pin);
00013     long frequency = 433E6;
00014     bool init_status = false;
00015     if (!LoRa.begin(frequency))
00016     {
00017         uart.print("LoRa init failed. Check your connections.", VerbosityLevel::WARNING);
00018         init_status = false;
00019     } else {
00020         uart.print("LoRa initialized with frequency " + std::to_string(frequency),
00021             VerbosityLevel::INFO);
00022         init_status = true;
00023     }
00024     EventEmitter::emit(EventGroup::COMMS, init_status ? CommsEvent::RADIO_INIT :
00025         CommsEvent::RADIO_ERROR);
00026
00027     return init_status;
00028 }
00029
00030
00031
00032
00033 }
00034

```

## 8.29 lib/comms/communication.h File Reference

```

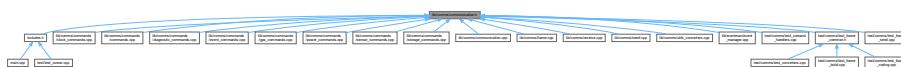
#include <string>
#include <vector>
#include "protocol.h"
#include "event_manager.h"

```

Include dependency graph for communication.h:



This graph shows which files directly or indirectly include this file:



### Functions

- `bool initialize_radio ()`  
*Initializes the LoRa radio module.*
- `void on_receive (int packetSize)`  
*Callback function for handling received LoRa packets.*
- `void handle_uart_input ()`

- Handles UART input.
- void `send_message` (std::string `outgoing`)
- void `send_frame_uart` (const `Frame` &frame)
- void `send_frame_lora` (const `Frame` &frame)
- void `split_and_send_message` (const uint8\_t \*data, size\_t length)
  - Sends a large packet using LoRa.*
- std::vector< `Frame` > `execute_command` (uint32\_t commandKey, const std::string &param, `OperationType` operationType)
  - Executes a command based on its key.*
- void `frame_process` (const std::string &data, `Interface` interface)
  - Executes a command based on the command key and the parameter.*
- std::string `frame_encode` (const `Frame` &frame)
  - Encodes a `Frame` instance into a string.*
- `Frame frame_decode` (const std::string &data)
  - Decodes a string into a `Frame` instance.*
- `Frame frame_build` (`OperationType` operation, uint8\_t group, uint8\_t command, const std::string &value, const `ValueUnit` unitType= `ValueUnit::UNDEFINED`)
  - Builds a `Frame` instance based on the execution result, group, command, value, and unit.*
- std::string `determine_unit` (uint8\_t group, uint8\_t command)

## 8.29.1 Function Documentation

### 8.29.1.1 initialize\_radio()

```
bool initialize_radio ()
```

Initializes the LoRa radio module.

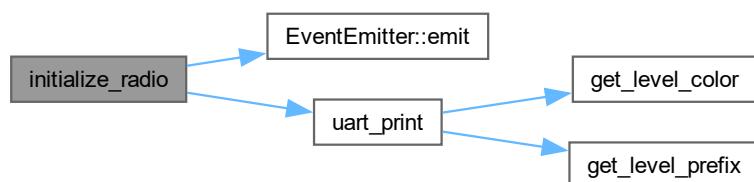
#### Returns

True if initialization was successful, false otherwise.

Sets the LoRa pins and attempts to begin LoRa communication at a specified frequency. Emits a `CommsEvent::RADIO_INIT` event on success or a `CommsEvent::RADIO_ERROR` event on failure.

Definition at line 17 of file `communication.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.29.1.2 on\_receive()

```
void on_receive (
    int packet_size)
```

Callback function for handling received LoRa packets.

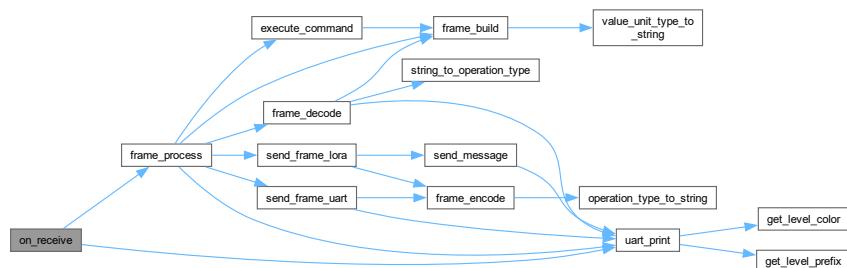
#### Parameters

<i>packet_size</i>	The size of the received packet.
--------------------	----------------------------------

Reads the received LoRa packet, extracts metadata, validates the lora\_address\_remote and local addresses, extracts the frame data, and processes it. Prints raw hex values for debugging.

Definition at line 15 of file [receive.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.29.1.3 handle\_uart\_input()

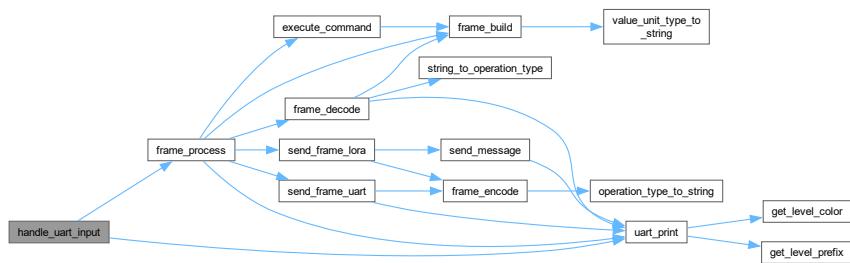
```
void handle_uart_input ()
```

Handles UART input.

Reads characters from the UART port, appends them to a buffer, and processes the buffer when a newline character is received.

Definition at line 76 of file [receive.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.29.1.4 send\_message()

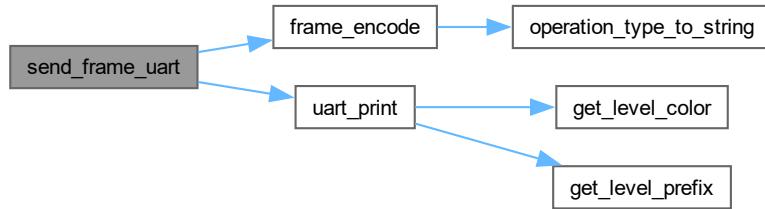
```
void send_message (
    std::string outgoing)
```

### 8.29.1.5 send\_frame\_uart()

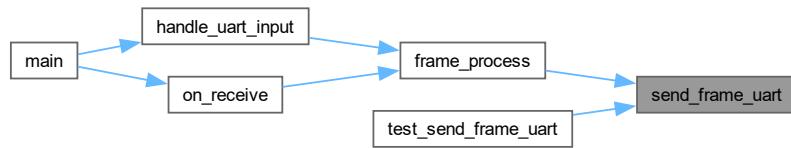
```
void send_frame_uart (
    const Frame & frame)
```

Definition at line 42 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

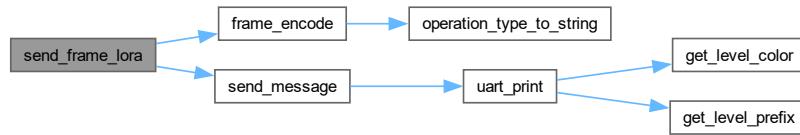


### 8.29.1.6 `send_frame_lora()`

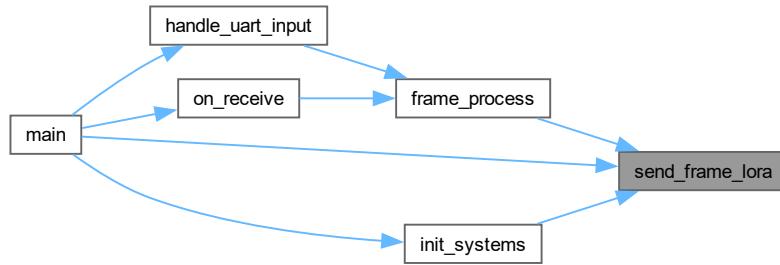
```
void send_frame_lora (
    const Frame & frame)
```

Definition at line 37 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.29.1.7 split\_and\_send\_message()

```
void split_and_send_message (
    const uint8_t * data,
    size_t length)
```

Sends a large packet using LoRa.

#### Parameters

<code>data</code>	The data to send.
<code>length</code>	The length of the data.

Splits the data into chunks of MAX\_PKT\_SIZE and sends each chunk as a separate LoRa packet.

Definition at line 54 of file [send.cpp](#).

### 8.29.1.8 determine\_unit()

```
std::string determine_unit (
    uint8_t group,
    uint8_t command)
```

## 8.30 communication.h

[Go to the documentation of this file.](#)

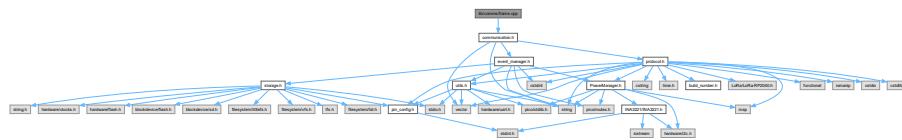
```
00001 #ifndef COMMUNICATION_H
00002 #define COMMUNICATION_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include "protocol.h"
00007 #include "event_manager.h"
00008
00009 bool initialize_radio();
00010 void on_receive(int packetSize);
```

```
00011 void handle_uart_input();
00012 void send_message(std::string outgoing);
00013 void send_frame_uart(const Frame& frame);
00014 void send_frame_lora(const Frame& frame);
00015
00016 void split_and_send_message(const uint8_t* data, size_t length);
00017
00018 std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType
operationType);
00019
00020 void frame_process(const std::string& data, Interface interface);
00021 std::string frame_encode(const Frame& frame);
00022 Frame frame_decode(const std::string& data);
00023 Frame frame_build(OperationType operation, uint8_t group, uint8_t command,const std::string& value,
const ValueUnit unitType = ValueUnit::UNDEFINED);
00024
00025 std::string determine_unit(uint8_t group, uint8_t command);
00026
00027 #endif
```

## 8.31 lib/comms/frame.cpp File Reference

Implements functions for encoding, decoding, building, and processing Frames.

```
#include "communication.h"
Include dependency graph for frame.cpp:
```



## TypeDefs

- using CommandHandler = std::function<std::vector<Frame>(&const std::string, OperationType)>;

## Functions

- `std::string frame_encode (const Frame &frame)`  
*Encodes a `Frame` instance into a string.*
  - `Frame frame_decode (const std::string &data)`  
*Decodes a string into a `Frame` instance.*
  - `void frame_process (const std::string &data, Interface interface)`  
*Executes a command based on the command key and the parameter.*
  - `Frame frame_build (OperationType operation, uint8_t group, uint8_t command, const std::string &value, const ValueUnit unitType)`  
*Builds a `Frame` instance based on the execution result, group, command, value, and unit.*

## Variables

- `std::map< uint32_t, CommandHandler > command_handlers`  
*Global map of all command handlers.*
  - `volatile uint16_t eventRegister`

### 8.31.1 Detailed Description

Implements functions for encoding, decoding, building, and processing Frames.

Definition in file [frame.cpp](#).

### 8.31.2 Typedef Documentation

#### 8.31.2.1 CommandHandler

```
using CommandHandler = std::function<std::vector<Frame>(const std::string&, OperationType)>
```

Definition at line 3 of file [frame.cpp](#).

### 8.31.3 Variable Documentation

#### 8.31.3.1 eventRegister

```
volatile uint16_t eventRegister [extern]
```

## 8.32 frame.cpp

[Go to the documentation of this file.](#)

```
00001 #include "communication.h"
00002
00003 using CommandHandler = std::function<std::vector<Frame>(const std::string&, OperationType)>;
00004 extern std::map<uint32_t, CommandHandler> command_handlers;
00005 extern volatile uint16_t eventRegister;
00006
00014
00037 std::string frame_encode(const Frame& frame) {
00038     std::stringstream ss;
00039     ss << static_cast<int>(frame.direction) << DELIMITER
00040         << operation_type_to_string(frame.operationType) << DELIMITER
00041         << static_cast<int>(frame.group) << DELIMITER
00042         << static_cast<int>(frame.command) << DELIMITER
00043         << frame.value;
00044
00045     if (!frame.unit.empty()) {
00046         ss << DELIMITER << frame.unit;
00047     }
00048
00049     return FRAME_BEGIN + DELIMITER + ss.str() + DELIMITER + FRAME_END;
00050 }
00051
00052
00062 Frame frame_decode(const std::string& data) {
00063     try {
00064         Frame frame;
00065         std::stringstream ss(data);
00066         std::string token;
00067
00068         std::getline(ss, token, DELIMITER);
00069         if (token != FRAME_BEGIN)
00070             throw std::runtime_error("Invalid frame header");
00071         frame.header = token;
00072
00073         std::string decoded_frame_data;
00074         while (std::getline(ss, token, DELIMITER)) {
00075             if (token == FRAME_END) break;
00076             decoded_frame_data += token + DELIMITER;
00077         }
00078         if (!decoded_frame_data.empty())
00079             decoded_frame_data.pop_back();
```

```

00080         }
00081
00082         std::stringstream frame_data_stream(decoded_frame_data);
00083
00084         std::getline(frame_data_stream, token, DELIMITER);
00085         frame.direction = std::stoi(token);
00086
00087         std::getline(frame_data_stream, token, DELIMITER);
00088         frame.operationType = string_to_operation_type(token);
00089
00090         std::getline(frame_data_stream, token, DELIMITER);
00091         frame.group = std::stoi(token);
00092
00093         std::getline(frame_data_stream, token, DELIMITER);
00094         frame.command = std::stoi(token);
00095
00096         std::getline(frame_data_stream, token, DELIMITER);
00097         frame.value = token;
00098
00099         std::getline(frame_data_stream, token, DELIMITER);
00100         frame.unit = token;
00101
00102         return frame;
00103     } catch (const std::exception& e) {
00104         uart_print("Frame error: " + std::string(e.what()), VerbosityLevel::ERROR);
00105         Frame error_frame = frame_build(OperationType::ERR, 0, 0, e.what());
00106         return error_frame;
00107     }
00108 }
00109
00110
00111 00117 void frame_process(const std::string& data, Interface interface) {
00112     gpio_put(PICO_DEFAULT_LED_PIN, 0);
00113
00114     uart_print("Processing frame: " + data, VerbosityLevel::WARNING);
00115     try {
00116         Frame frame = frame_decode(data);
00117         uint32_t command_key = (static_cast<uint32_t>(frame.group) << 8) |
00118             static_cast<uint32_t>(frame.command);
00119
00120         std::vector<Frame> response_frames = execute_command(command_key, frame.value,
00121             frame.operationType);
00122
00123         gpio_put(PICO_DEFAULT_LED_PIN, 1);
00124
00125         // Send all responses through the same interface that received the command
00126         for (const auto& response_frame : response_frames) {
00127             if (interface == Interface::UART) {
00128                 send_frame_uart(response_frame);
00129             } else if (interface == Interface::LORA) {
00130                 send_frame_lora(response_frame);
00131                 sleep_ms(50);
00132             }
00133         }
00134     } catch (const std::exception& e) {
00135         Frame error_frame = frame_build(OperationType::ERR, 0, 0, e.what());
00136         if (interface == Interface::UART) {
00137             send_frame_uart(error_frame);
00138         } else if (interface == Interface::LORA) {
00139             send_frame_lora(error_frame);
00140         }
00141     }
00142 }
00143
00144
00145 00158 Frame frame_build(OperationType operation, uint8_t group, uint8_t command,
00156         const std::string& value, const ValueUnit unitType) {
00157     Frame frame;
00158     frame.header = FRAME_BEGIN;
00159     frame.footer = FRAME_END;
00160
00161     switch (operation) {
00162         case OperationType::VAL:
00163             frame.direction = 1;
00164             frame.operationType = OperationType::VAL;
00165             frame.value = value;
00166             frame.unit = value_unit_type_to_string(unitType);
00167             break;
00168
00169         case OperationType::ERR:
00170             frame.direction = 1;
00171             frame.operationType = OperationType::ERR;
00172             frame.value = value;
00173             frame.unit = value_unit_type_to_string(ValueUnit::UNDEFINED);
00174             break;
00175
00176         case OperationType::RES:
00177             frame.direction = 1;
00178
00179     }
00180 }
```

```

00181     frame.operationType = OperationType::RES;
00182     frame.value = value;
00183     frame.unit = value_unit_type_to_string(unitType);
00184     break;
00185
00186     case OperationType::SEQ:
00187         frame.direction = 1;
00188         frame.operationType = OperationType::SEQ;
00189         frame.value = value;
00190         frame.unit = value_unit_type_to_string(unitType);
00191         break;
00192     }
00193
00194     frame.group = group;
00195     frame.command = command;
00196
00197     return frame;
00198 }
00199 // end of FrameHandling group

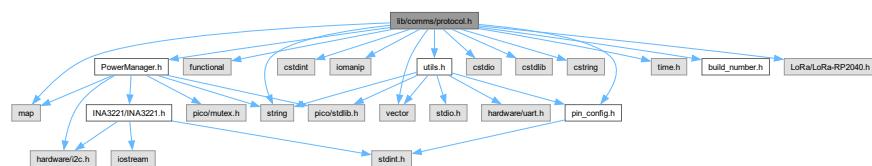
```

## 8.33 lib/comms/protocol.h File Reference

```

#include <string>
#include <map>
#include <functional>
#include <vector>
#include <cstdint>
#include <iomanip>
#include "pin_config.h"
#include "PowerManager.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include "utils.h"
#include "time.h"
#include "build_number.h"
#include "LoRa/LoRa-RP2040.h"
Include dependency graph for protocol.h:

```



This graph shows which files directly or indirectly include this file:



## Classes

- struct `Frame`

*Represents a communication frame used for data exchange.*

## Enumerations

- enum class `ErrorCode` {
   
    `PARAM_UNNECESSARY` , `PARAM_REQUIRED` , `PARAM_INVALID` , `INVALID_OPERATION` ,
   
    `NOT_ALLOWED` , `INVALID_FORMAT` , `INVALID_VALUE` , `FAIL_TO_SET` ,
   
    `INTERNAL_FAIL_TO_READ` , `UNKNOWN_ERROR` }
   
    *Standard error codes for command responses.*
- enum class `OperationType` {
   
    `GET` , `SET` , `RES` , `VAL` ,
   
    `SEQ` , `ERR` }
   
    *Represents the type of operation being performed.*
- enum class `CommandAccessLevel` { `NONE` , `READ_ONLY` , `WRITE_ONLY` , `READ_WRITE` }
   
    *Represents the access level required to execute a command.*
- enum class `ValueUnit` {
   
    `UNDEFINED` , `SECOND` , `VOLT` , `BOOL` ,
   
    `DATETIME` , `TEXT` , `MILIAMP` }
   
    *Represents the unit of measurement for a payload value.*
- enum class `ExceptionType` {
   
    `NONE` , `NOT_ALLOWED` , `INVALID_PARAM` , `INVALID_OPERATION` ,
   
    `PARAM_UNNECESSARY` }
   
    *Represents the type of exception that occurred during command execution.*
- enum class `Interface` { `UART` , `LORA` }
   
    *Represents the communication interface being used.*

## Functions

- std::string `exception_type_to_string` (`ExceptionType` type)
   
    *Converts an `ExceptionType` to a string.*
- std::string `error_code_to_string` (`ErrorCode` code)
   
    *Converts an `ErrorCode` to its string representation.*
- std::string `operation_type_to_string` (`OperationType` type)
   
    *Converts an `OperationType` to a string.*
- `OperationType` `string_to_operation_type` (const std::string &str)
   
    *Converts a string to an `OperationType`.*
- std::vector< uint8\_t > `hex_string_to_bytes` (const std::string &hexString)
   
    *Converts a hex string to a vector of bytes.*
- std::string `value_unit_type_to_string` (`ValueUnit` unit)
   
    *Converts a `ValueUnit` to a string.*

## Variables

- const std::string `FRAME_BEGIN` = "KBST"
- const std::string `FRAME_END` = "TSBK"
- const char `DELIMITER` = ';'

### 8.33.1 Enumeration Type Documentation

#### 8.33.1.1 ErrorCode

```
enum class ErrorCode [strong]
```

Standard error codes for command responses.

**Enumerator**

PARAM_UNNECESSARY	
PARAM_REQUIRED	
PARAM_INVALID	
INVALID_OPERATION	
NOT_ALLOWED	
INVALID_FORMAT	
INVALID_VALUE	
FAIL_TO_SET	
INTERNAL_FAIL_TO_READ	
UNKNOWN_ERROR	

Definition at line 45 of file [protocol.h](#).

**8.33.1.2 OperationType**

```
enum class OperationType [strong]
```

Represents the type of operation being performed.

**Enumerator**

GET	Get data.
SET	Set data.
RES	Set command result.
VAL	Get command value.
SEQ	Sequence element response.
ERR	Error occurred during command execution.

Definition at line 63 of file [protocol.h](#).

**8.33.1.3 CommandAccessLevel**

```
enum class CommandAccessLevel [strong]
```

Represents the access level required to execute a command.

**Enumerator**

NONE	No access allowed.
READ_ONLY	Read-only access.
WRITE_ONLY	Write-only access.
READ_WRITE	Read and write access.

Definition at line 85 of file [protocol.h](#).

**8.33.1.4 ValueUnit**

```
enum class ValueUnit [strong]
```

Represents the unit of measurement for a payload value.

**Enumerator**

UNDEFINED	Unit is undefined.
SECOND	Unit is seconds.
VOLT	Unit is volts.
BOOL	Unit is boolean.
DATETIME	Unit is date and time.
TEXT	Unit is text.
MILIAMP	Unit is milliamperes.

Definition at line 102 of file [protocol.h](#).

**8.33.1.5 ExceptionType**

```
enum class ExceptionType [strong]
```

Represents the type of exception that occurred during command execution.

**Enumerator**

NONE	No exception.
NOT_ALLOWED	Operation not allowed.
INVALID_PARAM	Invalid parameter provided.
INVALID_OPERATION	Invalid operation requested.
PARAM_UNNECESSARY	Parameter is unnecessary for the operation.

Definition at line 125 of file [protocol.h](#).

**8.33.1.6 Interface**

```
enum class Interface [strong]
```

Represents the communication interface being used.

**Enumerator**

UART	UART interface.
LORA	LoRa interface.

Definition at line 144 of file [protocol.h](#).

**8.33.2 Function Documentation****8.33.2.1 exception\_type\_to\_string()**

```
std::string exception_type_to_string (
    ExceptionType type)
```

Converts an [ExceptionType](#) to a string.

**Parameters**

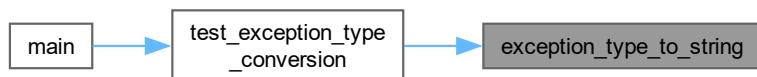
<i>type</i>	The <a href="#">ExceptionType</a> to convert.
-------------	---

**Returns**

The string representation of the [ExceptionType](#).

Definition at line 14 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.33.2.2 error\_code\_to\_string()

```
std::string error_code_to_string ( ErrorCode code)
```

Converts an [ErrorCode](#) to its string representation.

**Parameters**

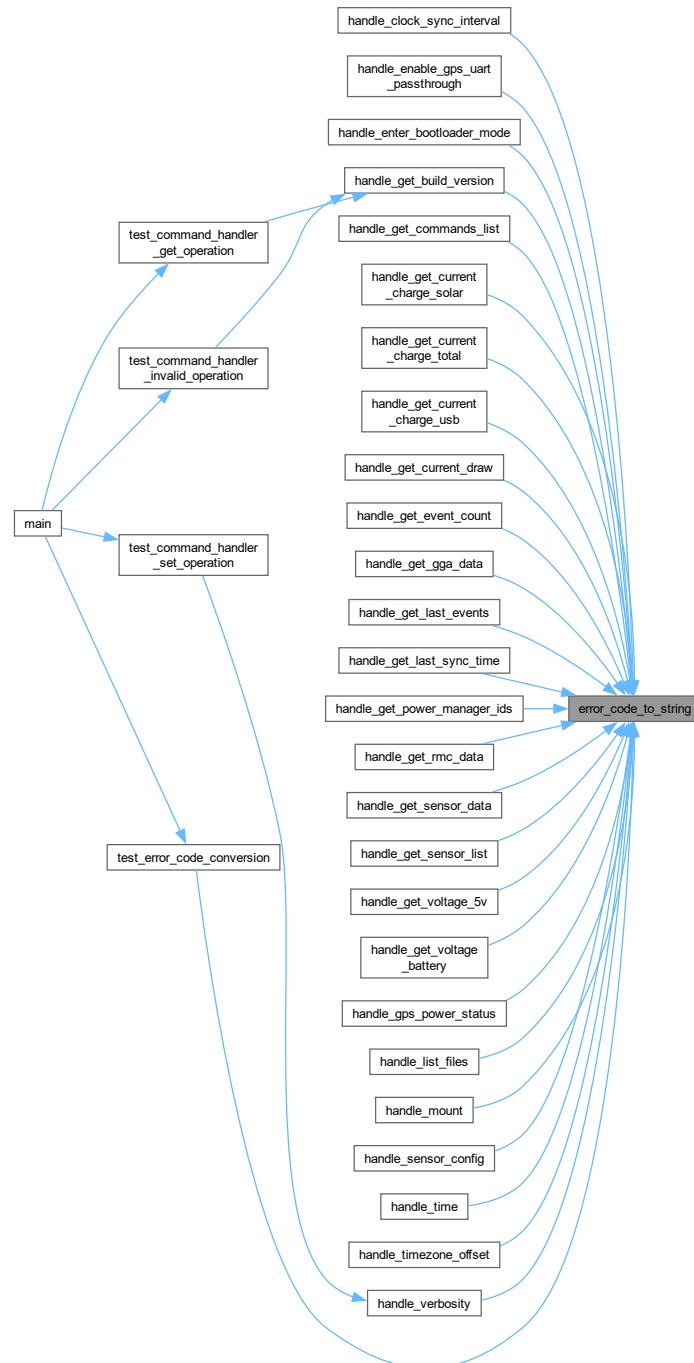
<i>code</i>	The error code
-------------	----------------

**Returns**

String representation of the error code

Definition at line 83 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.33.2.3 `operation_type_to_string()`

```
std::string operation_type_to_string (
    OperationType type)
```

Converts an `OperationType` to a string.

**Parameters**

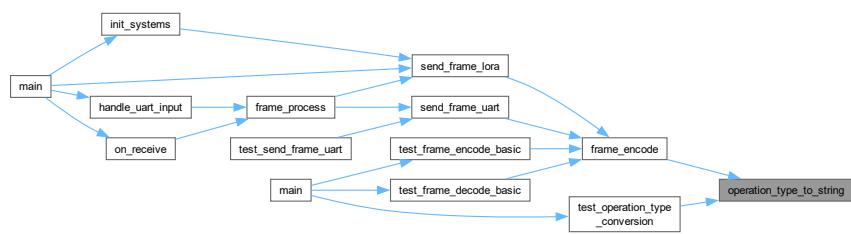
<code>type</code>	The <code>OperationType</code> to convert.
-------------------	--

**Returns**

The string representation of the `OperationType`.

Definition at line 50 of file `utils_converters.cpp`.

Here is the caller graph for this function:

**8.33.2.4 string\_to\_operation\_type()**

```
OperationType string_to_operation_type (
    const std::string & str)
```

Converts a string to an `OperationType`.

**Parameters**

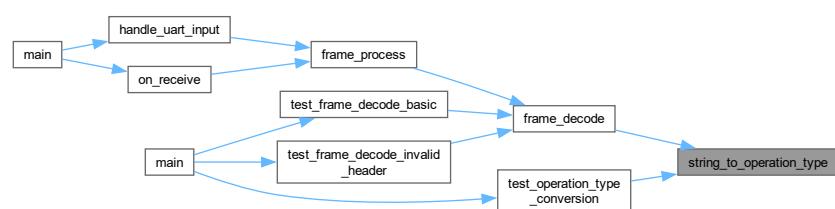
<code>str</code>	The string to convert.
------------------	------------------------

**Returns**

The `OperationType` corresponding to the string. Defaults to GET if the string is not recognized.

Definition at line 68 of file `utils_converters.cpp`.

Here is the caller graph for this function:



### 8.33.2.5 hex\_string\_to\_bytes()

```
std::vector< uint8_t > hex_string_to_bytes (
    const std::string & hexString)
```

Converts a hex string to a vector of bytes.

#### Parameters

<i>hexString</i>	The hex string to convert.
------------------	----------------------------

#### Returns

A vector of bytes representing the hex string.

Definition at line 104 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.33.2.6 value\_unit\_type\_to\_string()

```
std::string value_unit_type_to_string (
    ValueUnit unit)
```

Converts a [ValueUnit](#) to a string.

#### Parameters

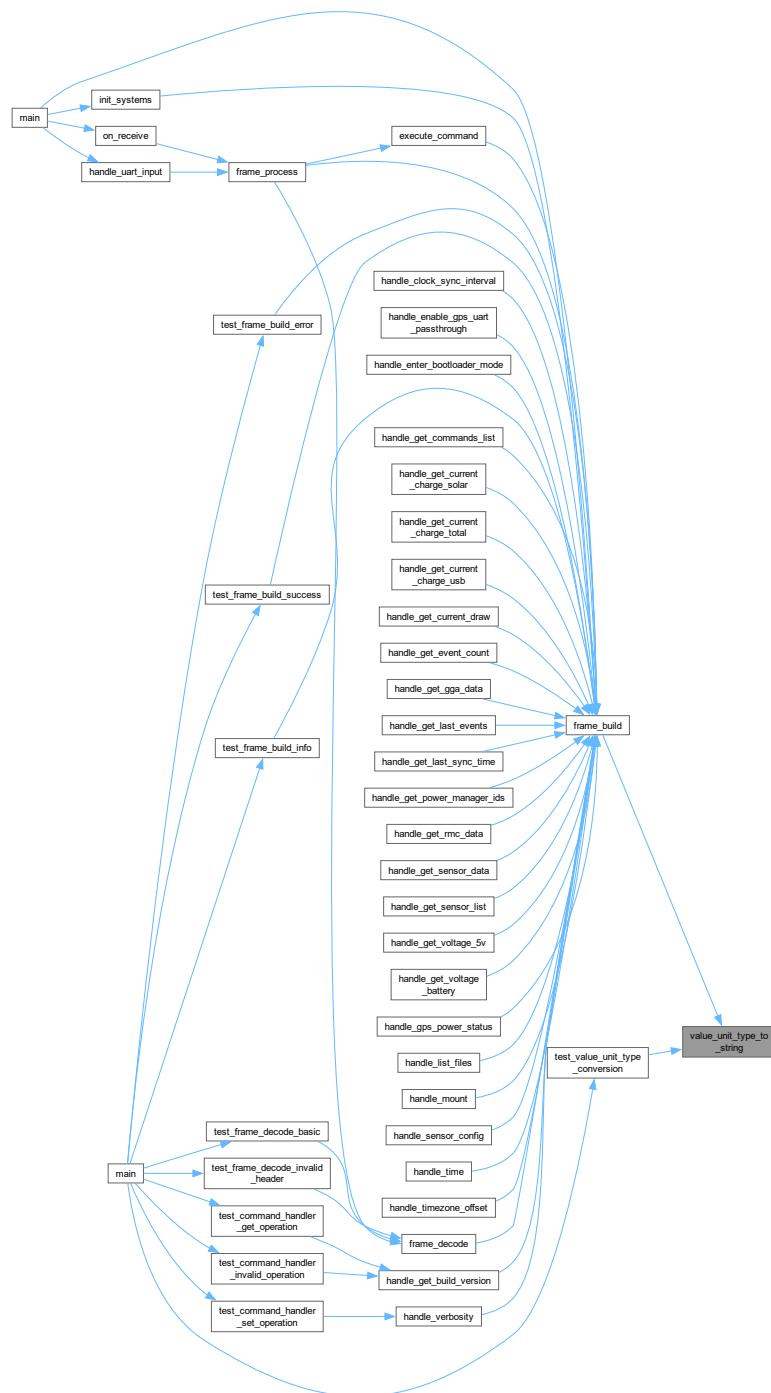
<i>unit</i>	The <a href="#">ValueUnit</a> to convert.
-------------	---

**Returns**

The string representation of the [ValueUnit](#).

Definition at line 31 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.33.3 Variable Documentation

#### 8.33.3.1 FRAME\_BEGIN

```
const std::string FRAME_BEGIN = "KBST"
```

Definition at line 26 of file [protocol.h](#).

#### 8.33.3.2 FRAME\_END

```
const std::string FRAME_END = "TSBK"
```

Definition at line 32 of file [protocol.h](#).

#### 8.33.3.3 DELIMITER

```
const char DELIMITER = ';'
```

Definition at line 38 of file [protocol.h](#).

## 8.34 protocol.h

[Go to the documentation of this file.](#)

```
00001 // protocol.h
00002 #ifndef PROTOCOL_H
00003 #define PROTOCOL_H
00004
00005 #include <string>
00006 #include <map>
00007 #include <functional>
00008 #include <vector>
00009 #include <cstdint>
00010 #include <iomanip>
00011 #include "pin_config.h"
00012 #include "PowerManager.h"
00013 #include <cstdio>
00014 #include <cstdlib>
00015 #include <map>
00016 #include <cstring>
00017 #include "utils.h"
00018 #include "time.h"
00019 #include "build_number.h"
00020 #include "LoRa/LoRa-RP2040.h"
00021
00026 const std::string FRAME_BEGIN = "KBST";
00027
00032 const std::string FRAME_END = "TSBK";
00033
00038 const char DELIMITER = ';';
00039
00040
00045 enum class ErrorCode {
00046     PARAM_UNNECESSARY,           // Parameter provided but not needed
00047     PARAM_REQUIRED,              // Required parameter missing
00048     PARAM_INVALID,               // Parameter has invalid format or value
00049     INVALID_OPERATION,           // Operation not allowed for this command
00050     NOT_ALLOWED,                 // Operation not permitted
00051     INVALID_FORMAT,              // Input format is incorrect
00052     INVALID_VALUE,                // Value is outside expected range
00053     FAIL_TO_SET,                  // Failed to set provided value
00054     INTERNAL_FAIL_TO_READ,        // Failed to read from device in remote
00055     UNKNOWN_ERROR                 // Generic error
00056 };
00057
00058
```

```

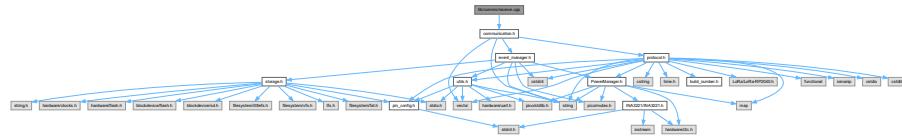
00063 enum class OperationType {
00065     GET,
00067     SET,
00069     RES,
00071     VAL,
00073     SEQ,
00075     ERR,
00076 };
00078
00079
00080
00085 enum class CommandAccessLevel {
00087     NONE,
00089     READ_ONLY,
00091     WRITE_ONLY,
00093     READ_WRITE
00094 };
00095
00096
00097
00102 enum class ValueUnit {
00104     UNDEFINED,
00106     SECOND,
00108     VOLT,
00110     BOOL,
00112     DATETIME,
00114     TEXT,
00116     MILIAMP,
00117 };
00118
00119
00120
00125 enum class ExceptionType {
00127     NONE,
00129     NOT_ALLOWED,
00131     INVALID_PARAM,
00133     INVALID_OPERATION,
00135     PARAM_UNNECESSARY
00136 };
00137
00138
00139
00144 enum class Interface {
00146     UART,
00148     LORA
00149 };
00150
00151
00227 struct Frame {
00228     std::string header;           // Start marker
00229     uint8_t direction;          // 0 = ground->sat, 1 = sat->ground
00230     OperationType operationType;
00231     uint8_t group;              // Group ID
00232     uint8_t command;            // Command ID within group
00233     std::string value;          // Payload value
00234     std::string unit;           // Payload unit
00235     std::string footer;         // End marker
00236 };
00237
00238 std::string exception_type_to_string(ExceptionType type);
00239 std::string error_code_to_string(ErrorCode code);
00240 std::string operation_type_to_string(OperationType type);
00241 OperationType string_to_operation_type(const std::string& str);
00242 std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString);
00243 std::string value_unit_type_to_string(ValueUnit unit);
00244
00245 #endif

```

## 8.35 lib/comms/receive.cpp File Reference

Implements functions for receiving and processing data, including LoRa and UART input.

```
#include "communication.h"
Include dependency graph for receive.cpp:
```



## Functions

- void `on_receive (int packet_size)`  
*Callback function for handling received LoRa packets.*
- void `handle_uart_input ()`  
*Handles UART input.*

### 8.35.1 Detailed Description

Implements functions for receiving and processing data, including LoRa and UART input.

Definition in file [receive.cpp](#).

### 8.35.2 Function Documentation

#### 8.35.2.1 on\_receive()

```
void on_receive (
    int packet_size)
```

Callback function for handling received LoRa packets.

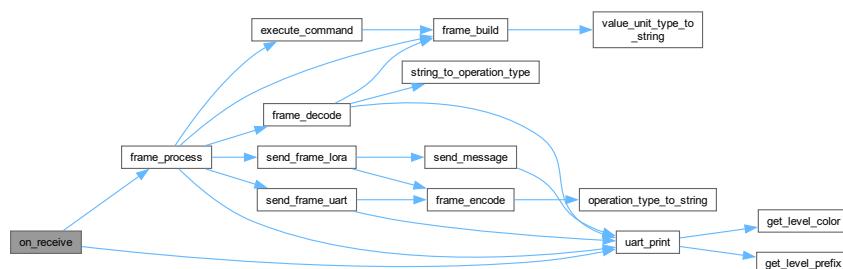
##### Parameters

<code>packet_size</code>	The size of the received packet.
--------------------------	----------------------------------

Reads the received LoRa packet, extracts metadata, validates the `lora_address_remote` and local addresses, extracts the frame data, and processes it. Prints raw hex values for debugging.

Definition at line 15 of file [receive.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.35.2.2 handle\_uart\_input()

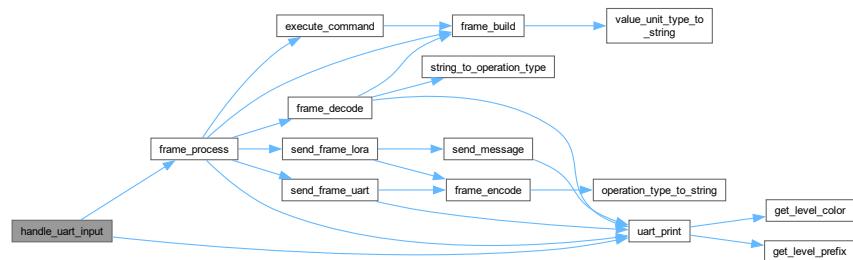
```
void handle_uart_input ()
```

Handles UART input.

Reads characters from the UART port, appends them to a buffer, and processes the buffer when a newline character is received.

Definition at line 76 of file [receive.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 8.36 receive.cpp

[Go to the documentation of this file.](#)

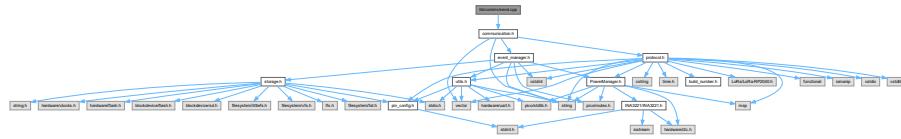
```

00001 #include "communication.h"
00002
00003
00008
00015 void on_receive(int packet_size) {
00016     if (packet_size == 0) return;
00017
00018     uint8_t buffer[256];
00019     int bytes_read = 0;
00020
00021     while (LoRa.available() && bytes_read < packet_size) {
00022         buffer[bytes_read++] = LoRa.read();
00023     }
00024
00025     // Extract LoRa metadata
00026     uint8_t received_destination = buffer[0];
00027     uint8_t received_local_address = buffer[1];
00028
00029     // Validate metadata (optional, for security)
00030     if (received_destination != lora_address_local) {
00031         uart_print("Error: Destination address mismatch!", VerbosityLevel::ERROR);
00032         return;
00033     }
00034
00035     if (received_local_address != lora_address_remote) {
00036         uart_print("Error: Local address mismatch!", VerbosityLevel::ERROR);
00037         return;
00038     }
00039
00040     // Find the starting index of the actual frame data
00041     int start_index = 2; // Start after the metadata
00042
00043     // Extract the frame data
00044     std::string received = std::string(reinterpret_cast<char*>(buffer + start_index), bytes_read -
00045     start_index);
00046
00047     if (received.empty()) return;
00048
00049     // Debug: Print raw hex values
00050     std::stringstream hex_dump;
00051     hex_dump << "Raw bytes: ";
00052     for (int i = 0; i < bytes_read; i++) {
00053         hex_dump << std::hex << std::setfill('0') << std::setw(2)
00054             << static_cast<int>(buffer[i]) << " ";
00055     }
00056     uart_print(hex_dump.str(), VerbosityLevel::DEBUG);
00057
00058     // Find frame boundaries
00059     size_t header_pos = received.find(FRAME_BEGIN);
00060     size_t footer_pos = received.find(FRAME_END);
00061
00062     if (header_pos != std::string::npos && footer_pos != std::string::npos && footer_pos > header_pos)
00063     {
00064         std::string frame_data = received.substr(header_pos, footer_pos + FRAME_END.length() -
00065         header_pos);
00066         uart_print("Extracted frame (length=" + std::to_string(frame_data.length()) + "): " +
00067         frame_data, VerbosityLevel::DEBUG);
00068         frame_process(frame_data, Interface::LORA);
00069     } else {
00070         uart_print("No valid frame found in received data", VerbosityLevel::WARNING);
00071     }
00072 }
00073
00074 void handle_uart_input() {
00075     static std::string uart_buffer;
00076
00077     while (uart_is_readable(DEBUG_UART_PORT)) {
00078         char c = uart_getc(DEBUG_UART_PORT);
00079
00080         if (c == '\r' || c == '\n') {
00081             uart_print("Received UART string: " + uart_buffer, VerbosityLevel::DEBUG);
00082             frame_process(uart_buffer, Interface::UART);
00083             uart_buffer.clear();
00084         } else {
00085             uart_buffer += c;
00086         }
00087     }
00088 }
00089 }
```

## 8.37 lib/comms/send.cpp File Reference

Implements functions for sending data, including LoRa messages and Frames.

```
#include "communication.h"
Include dependency graph for send.cpp:
```



### Functions

- `void send_message (string outgoing)`  
*Sends a message using LoRa.*
- `void send_frame_lora (const Frame &frame)`
- `void send_frame_uart (const Frame &frame)`
- `void split_and_send_message (const uint8_t *data, size_t length)`  
*Sends a large packet using LoRa.*

### 8.37.1 Detailed Description

Implements functions for sending data, including LoRa messages and Frames.

Definition in file [send.cpp](#).

### 8.37.2 Function Documentation

#### 8.37.2.1 send\_message()

```
void send_message (
    string outgoing)
```

Sends a message using LoRa.

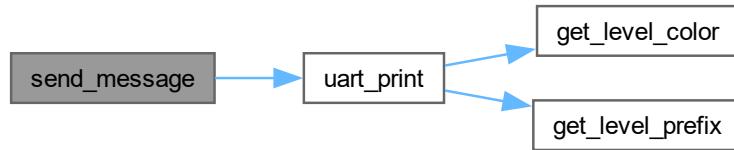
##### Parameters

<code>outgoing</code>	The message to send.
-----------------------	----------------------

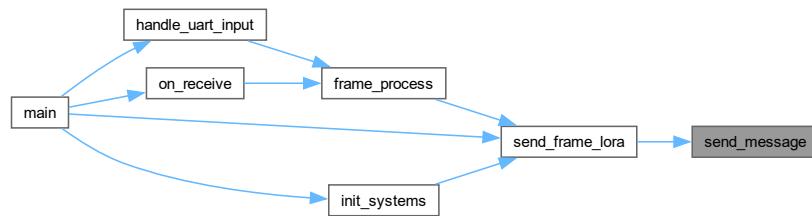
Converts the outgoing string to a C-style string, adds destination and local addresses, and sends the message using LoRa. Prints a log message to the UART.

Definition at line 15 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.37.2.2 send\_frame\_lora()

```
void send_frame_lora (
    const Frame & frame)
```

Definition at line 37 of file [send.cpp](#).

### 8.37.2.3 send\_frame\_uart()

```
void send_frame_uart (
    const Frame & frame)
```

Definition at line 42 of file [send.cpp](#).

### 8.37.2.4 split\_and\_send\_message()

```
void split_and_send_message (
    const uint8_t * data,
    size_t length)
```

Sends a large packet using LoRa.

**Parameters**

<i>data</i>	The data to send.
<i>length</i>	The length of the data.

Splits the data into chunks of MAX\_PKT\_SIZE and sends each chunk as a separate LoRa packet.

Definition at line 54 of file [send.cpp](#).

## 8.38 send.cpp

[Go to the documentation of this file.](#)

```

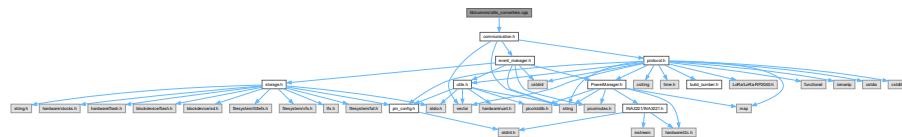
00001 #include "communication.h"
00002
00003
00008
00015 void send_message(string outgoing)
00016 {
00017     int n = outgoing.length();
00018     char send[n + 1];
00019     strcpy(send, outgoing.c_str());
00020
00021     LoRa.beginPacket();           // start packet
00022     LoRa.write(lora_address_remote); // add destination address
00023     LoRa.write(lora_address_local); // add sender address
00024     LoRa.print(send);           // add payload
00025     LoRa.endPacket(false);      // finish packet and send it
00026
00027     std::string message_to_log = "Sent message of size " + std::to_string(n);
00028     message_to_log += " to 0x" + std::to_string(lora_address_remote);
00029     message_to_log += " containing: " + string(send);
00030
00031     uart_print(message_to_log, VerbosityLevel::DEBUG);
00032
00033     LoRa.flush();
00034 }
00035
00036
00037 void send_frame_lora(const Frame& frame) {
00038     std::string encoded_frame = frame_encode(frame);
00039     send_message(encoded_frame);
00040 }
00041
00042 void send_frame_uart(const Frame& frame) {
00043     std::string encoded_frame = frame_encode(frame);
00044     uart_print(encoded_frame);
00045 }
00046
00047
00054 void split_and_send_message(const uint8_t* data, size_t length)
00055 {
00056     const size_t MAX_PKT_SIZE = 255;
00057     size_t offset = 0;
00058     while (offset < length)
00059     {
00060         size_t chunk_size = ((length - offset) < MAX_PKT_SIZE) ? (length - offset) : MAX_PKT_SIZE;
00061         LoRa.beginPacket();
00062         LoRa.write(&data[offset], chunk_size);
00063         LoRa.endPacket();
00064         offset += chunk_size;
00065         sleep_ms(100);
00066     }
00067 }
00068

```

## 8.39 lib/comms/utils\_converters.cpp File Reference

Implements utility functions for converting between different data types.

```
#include "communication.h"
Include dependency graph for utils_converters.cpp:
```



## Functions

- `std::string exception_type_to_string (ExceptionType type)`  
Converts an `ExceptionType` to a string.
- `std::string value_unit_type_to_string (ValueUnit unit)`  
Converts a `ValueUnit` to a string.
- `std::string operation_type_to_string (OperationType type)`  
Converts an `OperationType` to a string.
- `OperationType string_to_operation_type (const std::string &str)`  
Converts a string to an `OperationType`.
- `std::string error_code_to_string (ErrorCode code)`  
Converts an `ErrorCode` to its string representation.
- `std::vector< uint8_t > hex_string_to_bytes (const std::string &hexString)`  
Converts a hex string to a vector of bytes.

### 8.39.1 Detailed Description

Implements utility functions for converting between different data types.

Definition in file `utils_converters.cpp`.

### 8.39.2 Function Documentation

#### 8.39.2.1 exception\_type\_to\_string()

```
std::string exception_type_to_string (
    ExceptionType type)
```

Converts an `ExceptionType` to a string.

##### Parameters

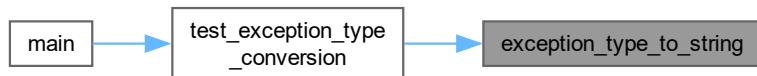
<code>type</code>	The <code>ExceptionType</code> to convert.
-------------------	--

**Returns**

The string representation of the [ExceptionType](#).

Definition at line 14 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.39.2.2 value\_unit\_type\_to\_string()

```
std::string value_unit_type_to_string (
    ValueUnit unit)
```

Converts a [ValueUnit](#) to a string.

**Parameters**

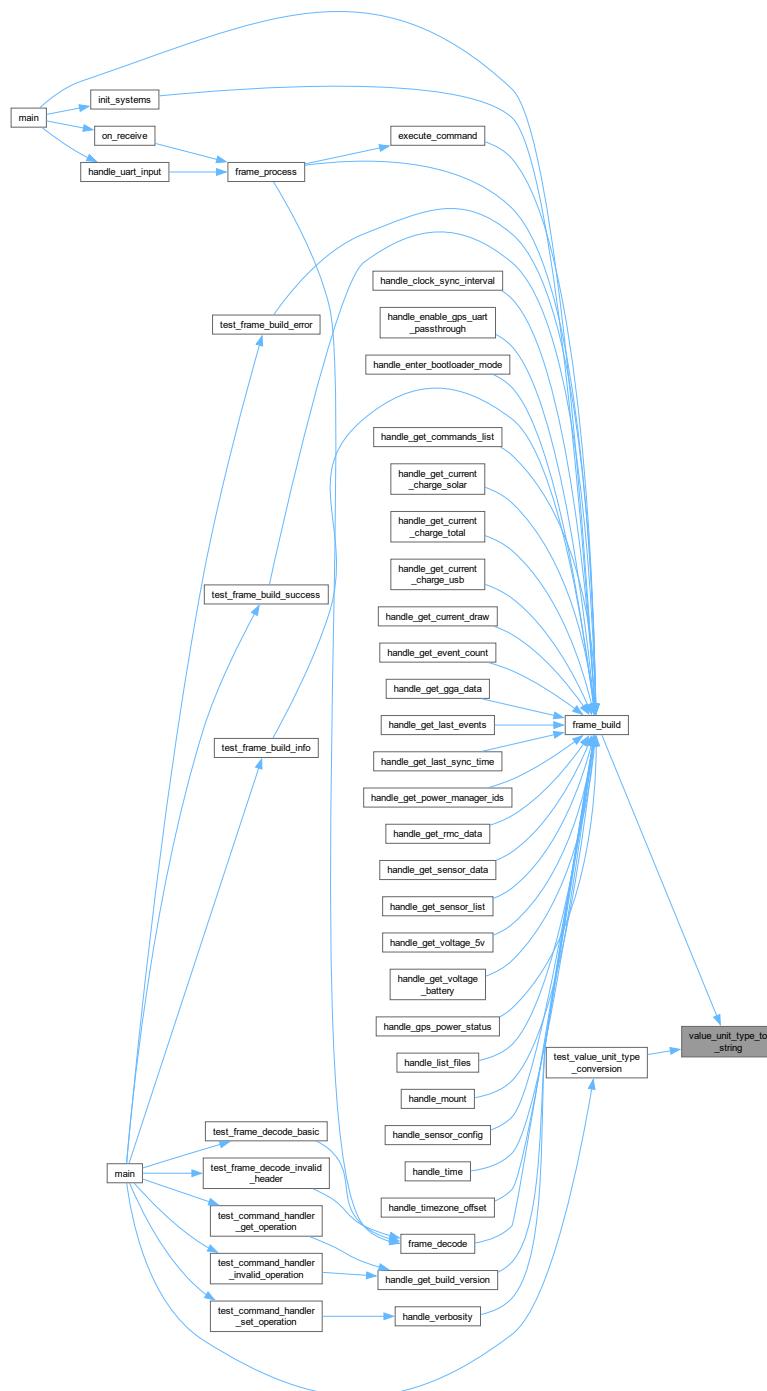
<i>unit</i>	The <a href="#">ValueUnit</a> to convert.
-------------	---

**Returns**

The string representation of the [ValueUnit](#).

Definition at line 31 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.39.2.3 operation\_type\_to\_string()

```
std::string operation_type_to_string (
    OperationType type)
```

Converts an `OperationType` to a string.

**Parameters**

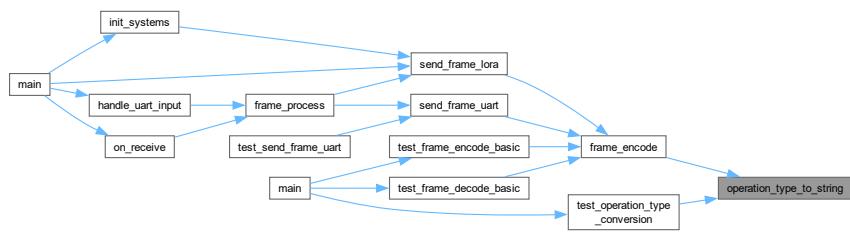
<code>type</code>	The <a href="#">OperationType</a> to convert.
-------------------	---

**Returns**

The string representation of the [OperationType](#).

Definition at line 50 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:

**8.39.2.4 string\_to\_operation\_type()**

```
OperationType string_to_operation_type (
    const std::string & str)
```

Converts a string to an [OperationType](#).

**Parameters**

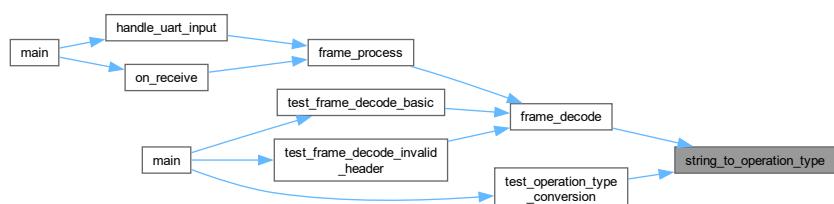
<code>str</code>	The string to convert.
------------------	------------------------

**Returns**

The [OperationType](#) corresponding to the string. Defaults to GET if the string is not recognized.

Definition at line 68 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.39.2.5 `error_code_to_string()`

```
std::string error_code_to_string (
    ErrorCode code)
```

Converts an `ErrorCode` to its string representation.

#### Parameters

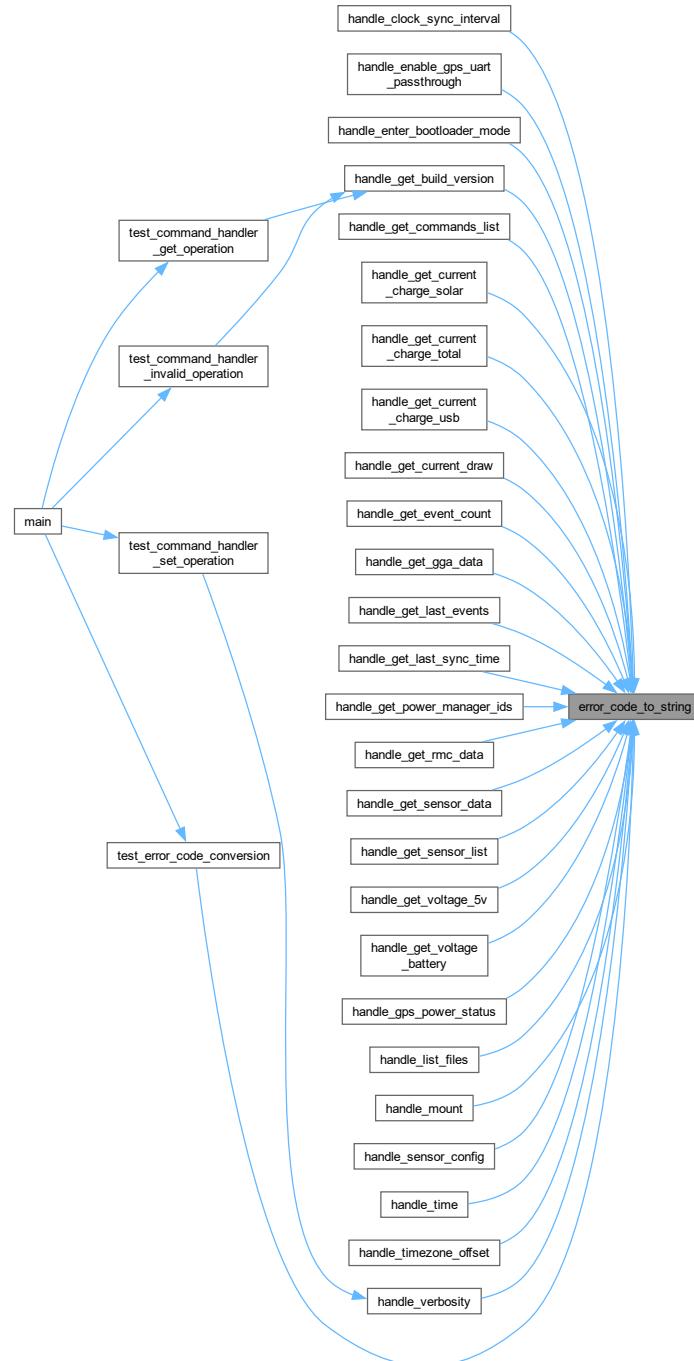
<code>code</code>	The error code
-------------------	----------------

#### Returns

String representation of the error code

Definition at line [83](#) of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



### 8.39.2.6 hex\_string\_to\_bytes()

```
std::vector< uint8_t > hex_string_to_bytes (
    const std::string & hexString)
```

Converts a hex string to a vector of bytes.

**Parameters**

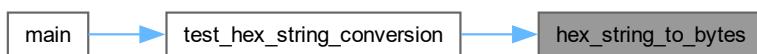
<i>hexString</i>	The hex string to convert.
------------------	----------------------------

**Returns**

A vector of bytes representing the hex string.

Definition at line 104 of file [utils\\_converters.cpp](#).

Here is the caller graph for this function:



## 8.40 utils\_converters.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002
00003
00008
00014 std::string exception_type_to_string(ExceptionType type) {
00015     switch (type) {
00016         case ExceptionType::NOT_ALLOWED:      return "NOT ALLOWED";
00017         case ExceptionType::INVALID_PARAM:    return "INVALID PARAM";
00018         case ExceptionType::INVALID_OPERATION: return "INVALID OPERATION";
00019         case ExceptionType::PARAM_UNNECESSARY: return "PARAM UNECESSARY";
00020         case ExceptionType::NONE:             return "NONE";
00021         default:                           return "UNKNOWN EXCEPTION";
00022     }
00023 }
00024
00025
00031 std::string value_unit_type_to_string(ValueUnit unit) {
00032     switch (unit) {
00033         case ValueUnit::UNDEFINED:   return "";
00034         case ValueUnit::SECOND:     return "s";
00035         case ValueUnit::VOLT:       return "V";
00036         case ValueUnit::BOOL:       return "";
00037         case ValueUnit::DATETIME:   return "";
00038         case ValueUnit::TEXT:       return "";
00039         case ValueUnit::MILLIAMP:   return "mA";
00040         default:                  return "";
00041     }
00042 }
00043
00044
00050 std::string operation_type_to_string(OperationType type) {
00051     switch (type) {
00052         case OperationType::GET:   return "GET";
00053         case OperationType::SET:   return "SET";
00054         case OperationType::VAL:   return "VAL";
00055         case OperationType::ERR:   return "ERR";
00056         case OperationType::RES:   return "RES";
00057         case OperationType::SEQ:   return "SEQ";
00058         default:                 return "UNKNOWN";
00059     }
00060 }
00061
00062
00068 OperationType string_to_operation_type(const std::string& str) {
00069     if (str == "GET") return OperationType::GET;

```

```
00070     if (str == "SET") return OperationType::SET;
00071     if (str == "VAL") return OperationType::VAL;
00072     if (str == "ERR") return OperationType::ERR;
00073     if (str == "RES") return OperationType::RES;
00074     if (str == "SEQ") return OperationType::SEQ;
00075     return OperationType::GET; // Default to GET
00076 }
00077
00083 std::string error_code_to_string(ErrorCode code) {
00084     switch (code) {
00085         case ErrorCode::PARAM_UNNECESSARY: return "PARAM_UNNECESSARY";
00086         case ErrorCode::PARAM_REQUIRED: return "PARAM_REQUIRED";
00087         case ErrorCode::PARAM_INVALID: return "PARAM_INVALID";
00088         case ErrorCode::INVALID_OPERATION: return "INVALID_OPERATION";
00089         case ErrorCode::NOT_ALLOWED: return "NOT_ALLOWED";
00090         case ErrorCode::INVALID_FORMAT: return "INVALID_FORMAT";
00091         case ErrorCode::INVALID_VALUE: return "INVALID_VALUE";
00092         case ErrorCode::FAIL_TO_SET: return "FAIL_TO_SET";
00093         case ErrorCode::INTERNAL_FAIL_TO_READ: return "INTERNAL_FAIL_TO_READ";
00094         default: return "UNKNOWN_ERROR";
00095     }
00096 }
00097
00098
00104 std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString) {
00105     std::vector<uint8_t> bytes;
00106     for (size_t i = 0; i < hexString.length(); i += 2) {
00107         std::string byteString = hexString.substr(i, 2);
00108         unsigned int byte;
00109         std::stringstream ss;
00110         ss << std::hex << byteString;
00111         ss >> byte;
00112         bytes.push_back(static_cast<uint8_t>(byte));
00113     }
00114     return bytes;
00115 }
```

## 8.41 lib/eventman/event\_manager.cpp File Reference

Implements the event management system for the Kubit firmware.

```
#include "event_manager.h"
#include <cstdio>
#include "protocol.h"
#include "pico/multicore.h"
#include "communication.h"
#include "utils.h"
#include "DS3231.h"

Include dependency graph for event_manager.cpp:
```

Include dependency graph for event\_manager.cpp:



## Variables

- volatile uint16\_t `eventLogId` = 0
    - Global event log ID counter.*
  - DS3231 `systemClock`
    - External declaration of the system clock.*
  - EventManagerImpl `eventManager`
    - Global instance of the `EventManager` implementation.*

### 8.41.1 Detailed Description

Implements the event management system for the Kabisat firmware.

This file contains the implementation for logging events, managing event storage, and checking for specific events such as power status changes.

Definition in file [event\\_manager.cpp](#).

## 8.42 event\_manager.cpp

[Go to the documentation of this file.](#)

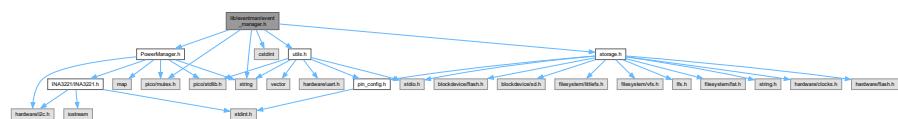
```

00001 #include "event_manager.h"
00002 #include <cstdio>
00003 #include "protocol.h"
00004 #include "pico/multicore.h"
00005 #include "communication.h"
00006 #include "utils.h"
00007 #include "DS3231.h"
00008
00016
00021 volatile uint16_t eventLogId = 0;
00022
00027 extern DS3231 systemClock;
00028
00033 EventManagerImpl eventManager;
00034
00035 uint16_t EventManager::nextEventId = 0;
00036
00045 void EventManager::log_event(uint8_t group, uint8_t event) {
00046     mutex_enter_blocking(&eventMutex);
00047
00048     // Clear buffer if it's full
00049     if (eventCount >= EVENT_BUFFER_SIZE) {
00050         eventCount = 0;
00051         writeIndex = 0;
00052     }
00053
00054     EventLog& log = events[writeIndex];
00055     log.id = nextEventId++;
00056     log.timestamp = systemClock.get_unix_time();
00057     log.group = group;
00058     log.event = event;
00059
00060     // Print event immediately
00061     uart_print(log.to_string(), VerbosityLevel::EVENT);
00062
00063     writeIndex = (writeIndex + 1) % EVENT_BUFFER_SIZE;
00064     eventCount++;
00065     eventsSinceFlush++;
00066
00067     // Flush to storage every EVENT_FLUSH_THRESHOLD events
00068     if (eventsSinceFlush >= EVENT_FLUSH_THRESHOLD) {
00069         save_to_storage();
00070         eventsSinceFlush = 0;
00071     }
00072
00073     mutex_exit(&eventMutex);
00074 }
00075
00076
00083 const EventLog& EventManager::get_event(size_t index) const {
00084     static const EventLog emptyEvent = {0, 0, 0, 0}; // Initialize {id, timestamp, group, event}
00085     if (index >= eventCount) {
00086         return emptyEvent;
00087     }
00088
00089     // Calculate actual index in circular buffer
00090     size_t actualIndex;
00091     if (eventCount == EVENT_BUFFER_SIZE) {
00092         actualIndex = (writeIndex + index) % EVENT_BUFFER_SIZE;
00093     } else {
00094         actualIndex = index;
00095     }
00096
00097     return events[actualIndex];
00098 }
```

## 8.43 lib/eventman/event\_manager.h File Reference

Manages the event logging system for the Kubisat firmware.

```
#include "PowerManager.h"
#include <cstdint>
#include <string>
#include "pico/mutex.h"
#include "storage.h"
#include "utils.h"
Include dependency graph for event_manager.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [EventLog](#)

*Represents a single event log entry.*
  - class [EventManager](#)

*Manages the event logging system.*
  - class [EventManagerImpl](#)

*Implementation of the [EventManager](#) class.*
  - class [EventEmitter](#)

*Provides a static method for emitting events.*

## Macros

- #define EVENT\_BUFFER\_SIZE 100
  - #define EVENT\_FLUSH\_THRESHOLD 10
  - #define EVENT\_LOG\_FILE "/event\_log.csv"

## Enumerations

- enum class `EventGroup` : uint8\_t {
   
  `EventGroup::SYSTEM` = 0x00 , `EventGroup::POWER` = 0x01 , `EventGroup::COMMS` = 0x02 ,
   
  `EventGroup::GPS` = 0x03 ,
   
  `EventGroup::CLOCK` = 0x04 }
   
*Represents the group to which an event belongs.*
- enum class `SystemEvent` : uint8\_t {
   
  `SystemEvent::BOOT` = 0x01 , `SystemEvent::SHUTDOWN` = 0x02 , `SystemEvent::WATCHDOG_RESET` =
   
  0x03 , `SystemEvent::CORE1_START` = 0x04 ,
   
  `SystemEvent::CORE1_STOP` = 0x05 }
   
*Represents specific system events.*
- enum class `PowerEvent` : uint8\_t {
   
  `PowerEvent::LOW_BATTERY` = 0x01 , `PowerEvent::OVERCHARGE` = 0x02 , `PowerEvent::POWER_FALLING`
= 0x03 , `PowerEvent::POWER_NORMAL` = 0x04 ,
   
  `PowerEvent::SOLAR_ACTIVE` = 0x05 , `PowerEvent::SOLAR_INACTIVE` = 0x06 , `PowerEvent::USB_CONNECTED`
= 0x07 , `PowerEvent::USB_DISCONNECTED` = 0x08 }
   
*Represents specific power-related events.*
- enum class `CommsEvent` : uint8\_t {
   
  `CommsEvent::RADIO_INIT` = 0x01 , `CommsEvent::RADIO_ERROR` = 0x02 , `CommsEvent::MSG_RECEIVED`
= 0x03 , `CommsEvent::MSG_SENT` = 0x04 ,
   
  `CommsEvent::UART_ERROR` = 0x06 }
   
*Represents specific communication-related events.*
- enum class `GPSEvent` : uint8\_t {
   
  `GPSEvent::LOCK` = 0x01 , `GPSEvent::LOST` = 0x02 , `GPSEvent::ERROR` = 0x03 , `GPSEvent::POWER_ON`
= 0x04 ,
   
  `GPSEvent::POWER_OFF` = 0x05 , `GPSEvent::DATA_READY` = 0x06 , `GPSEvent::PASS_THROUGH_START`
= 0x07 , `GPSEvent::PASS_THROUGH_END` = 0x08 }
   
*Represents specific GPS-related events.*
- enum class `ClockEvent` : uint8\_t { `ClockEvent::CHANGED` = 0x01 , `ClockEvent::GPS_SYNC` = 0x02 ,
   
`ClockEvent::GPS_SYNC_DATA_NOT_READY` = 0x03 }
   
*Represents specific clock-related events.*

## Functions

- class `EventLog __attribute__ ((packed))`
- std::string `to_string () const`

*Converts the `EventLog` to a string representation.*

## Variables

- uint16\_t `id`
  
*Sequence number.*
- uint32\_t `timestamp`
  
*Unix timestamp or system time.*
- uint8\_t `group`
  
*Event group identifier.*
- uint8\_t `event`
  
*Specific event identifier.*
- class `EventManager __attribute__`
- `EventManagerImpl eventManager`

*Global instance of the `EventManagerImpl` class.*

### 8.43.1 Detailed Description

Manages the event logging system for the KubitSat firmware.

Definition in file [event\\_manager.h](#).

### 8.43.2 Macro Definition Documentation

#### 8.43.2.1 EVENT\_BUFFER\_SIZE

```
#define EVENT_BUFFER_SIZE 100
```

Definition at line 11 of file [event\\_manager.h](#).

#### 8.43.2.2 EVENT\_FLUSH\_THRESHOLD

```
#define EVENT_FLUSH_THRESHOLD 10
```

Definition at line 12 of file [event\\_manager.h](#).

#### 8.43.2.3 EVENT\_LOG\_FILE

```
#define EVENT_LOG_FILE "/event_log.csv"
```

Definition at line 13 of file [event\\_manager.h](#).

### 8.43.3 Function Documentation

#### 8.43.3.1 to\_string()

```
std::string __attribute__::__to_string () const
```

Converts the [EventLog](#) to a string representation.

Returns

A string representation of the [EventLog](#).

Definition at line 14 of file [event\\_manager.h](#).

### 8.43.4 Variable Documentation

#### 8.43.4.1 id

```
uint16_t id
```

Sequence number.

Definition at line 2 of file [event\\_manager.h](#).

#### 8.43.4.2 timestamp

```
uint32_t timestamp
```

Unix timestamp or system time.

Definition at line 4 of file [event\\_manager.h](#).

#### 8.43.4.3 group

```
uint8_t group
```

Event group identifier.

Definition at line 6 of file [event\\_manager.h](#).

#### 8.43.4.4 event

```
uint8_t event
```

Specific event identifier.

Definition at line 8 of file [event\\_manager.h](#).

## 8.44 event\_manager.h

[Go to the documentation of this file.](#)

```
00001 #ifndef EVENT_MANAGER_H
00002 #define EVENT_MANAGER_H
00003
00004 #include "PowerManager.h"
00005 #include <cstdint>
00006 #include <string>
00007 #include "pico/mutex.h"
00008 #include "storage.h"
00009 #include "utils.h"
00010
00011 #define EVENT_BUFFER_SIZE 100
00012 #define EVENT_FLUSH_THRESHOLD 10
00013 #define EVENT_LOG_FILE "/event_log.csv"
00014
00022
00023
00028 enum class EventGroup : uint8_t {
00030     SYSTEM    = 0x00,
00032     POWER     = 0x01,
00034     COMMS     = 0x02,
00036     GPS       = 0x03,
00038     CLOCK     = 0x04
00039 };
00040
00045 enum class SystemEvent : uint8_t {
00047     BOOT       = 0x01,
00049     SHUTDOWN   = 0x02,
00051     WATCHDOG_RESET = 0x03,
00053     CORE1_START = 0x04,
00055     CORE1_STOP  = 0x05
00056 };
00057
00062 enum class PowerEvent : uint8_t {
00064     LOW_BATTERY    = 0x01,
00066     OVERCHARGE    = 0x02,
00068     POWER_FALLING = 0x03,
00070     POWER_NORMAL   = 0x04,
```

```
00072     SOLAR_ACTIVE      = 0x05,
00074     SOLAR_INACTIVE    = 0x06,
00076     USB_CONNECTED     = 0x07,
00078     USB_DISCONNECTED = 0x08
00079 };
00080
00085 enum class CommsEvent : uint8_t {
00087     RADIO_INIT        = 0x01,
00089     RADIO_ERROR       = 0x02,
00091     MSG_RECEIVED      = 0x03,
00093     MSG_SENT          = 0x04,
00095     UART_ERROR        = 0x06
00096 };
00097
00102 enum class GPSEvent : uint8_t {
00104     LOCK              = 0x01,
00106     LOST              = 0x02,
00108     ERROR             = 0x03,
00110     POWER_ON          = 0x04,
00112     POWER_OFF         = 0x05,
00114     DATA_READY        = 0x06,
00116     PASS_THROUGH_START = 0x07,
00118     PASS_THROUGH_END   = 0x08
00119 };
00120
00121
00126 enum class ClockEvent : uint8_t {
00128     CHANGED           = 0x01,
00130     GPS_SYNC          = 0x02,
00132     GPS_SYNC_DATA_NOT_READY = 0x03
00133 };
00134
00135
00140 class EventLog {
00141     public:
00143         uint16_t id;
00145         uint32_t timestamp;
00147         uint8_t group;
00149         uint8_t event;
00150
00155         std::string to_string() const {
00156             char buffer[256];
00157             snprintf(buffer, sizeof(buffer),
00158                     "EventLog: id=%u, timestamp=%lu, group=%u, event=%u",
00159                     id, timestamp, group, event);
00160             return std::string(buffer);
00161         }
00162     } __attribute__((packed));
00163
00164
00169 class EventManager {
00170     public:
00171         EventManager()
00172             : eventCount(0)
00173             , writeIndex(0)
00174             , eventsSinceFlush(0) // Add eventsSinceFlush initialization
00175         {
00176             mutex_init(&eventMutex);
00177         }
00178
00179         virtual ~EventManager() = default;
00180
00181         virtual void init() {
00182             load_from_storage();
00183         }
00184
00185         void log_event(uint8_t group, uint8_t event);
00186
00187         const EventLog& get_event(size_t index) const;
00188
00189         size_t get_event_count() const { return eventCount; }
00190
00191         virtual bool save_to_storage() = 0;
00192
00193         virtual bool load_from_storage() = 0;
00194
00195         protected:
00196             EventLog events[EVENT_BUFFER_SIZE];
00197             size_t eventCount;
00198             size_t writeIndex;
00199             mutex_t eventMutex;
00200             static uint16_t nextEventId;
00201             size_t eventsSinceFlush;
00202         };
00203
00204
00248 class EventManagerImpl : public EventManager {
```

```

00249     public:
00250         EventManagerImpl() {
00251             init(); // Safe to call virtual functions here
00252         }
00253
00254         public:
00255             bool save_to_storage() override {
00256                 if(!sd_card_mounted) {
00257                     bool status = fs_init();
00258                     if(!status) {
00259                         return false;
00260                     }
00261                 }
00262
00263                 FILE *file = fopen(EVENT_LOG_FILE, "a");
00264                 if (file) {
00265                     // Calculate start index for last EVENT_FLUSH_THRESHOLD events
00266                     size_t startIdx = (writeIndex >= eventsSinceFlush) ?
00267                         writeIndex - eventsSinceFlush :
00268                         EVENT_BUFFER_SIZE - (eventsSinceFlush - writeIndex);
00269
00270                     // Write only the most recent batch of events
00271                     for (size_t i = 0; i < eventsSinceFlush; i++) {
00272                         size_t idx = (startIdx + i) % EVENT_BUFFER_SIZE;
00273                         fprintf(file, "%u;%lu;%u;%u\n",
00274                             events[idx].id,
00275                             events[idx].timestamp,
00276                             events[idx].group,
00277                             events[idx].event
00278                         );
00279                     }
00280                     fclose(file);
00281                     uart_print("Events saved to storage", VerbosityLevel::INFO);
00282                     return true;
00283                 }
00284             }
00285         }
00286     };
00287
00288     extern EventManagerImpl eventManager;
00289
00301     class EventEmitter {
00302         public:
00303             template<typename T>
00304             static void emit(EventGroup group, T event) {
00305                 eventManager.log_event(
00306                     static_cast<uint8_t>(group),
00307                     static_cast<uint8_t>(event)
00308                 );
00309             }
00310     };
00311 #endif // End of EventManagerGroup

```

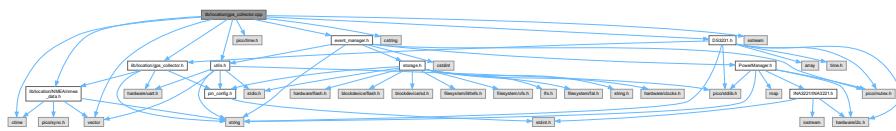
## 8.45 lib/location/gps\_collector.cpp File Reference

```

#include "lib/location/gps_collector.h"
#include "utils.h"
#include "pico/time.h"
#include "lib/location/NMEA/nmea_data.h"
#include "event_manager.h"
#include <vector>
#include <ctime>
#include <cstring>
#include "DS3231.h"
#include <iostream>

```

Include dependency graph for gps\_collector.cpp:



## Macros

- `#define MAX_RAW_DATA_LENGTH 256`

## Functions

- `std::vector< std::string > splitString (const std::string &str, char delimiter)`
- `void collect_gps_data ()`

## Variables

- `volatile bool pause_gps_collection`
- `NMEAData nmea_data`

### 8.45.1 Macro Definition Documentation

#### 8.45.1.1 MAX\_RAW\_DATA\_LENGTH

```
#define MAX_RAW_DATA_LENGTH 256
```

Definition at line 15 of file [gps\\_collector.cpp](#).

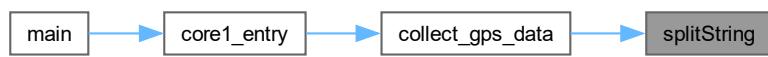
### 8.45.2 Function Documentation

#### 8.45.2.1 splitString()

```
std::vector< std::string > splitString (
    const std::string & str,
    char delimiter)
```

Definition at line 19 of file [gps\\_collector.cpp](#).

Here is the caller graph for this function:



#### 8.45.2.2 collect\_gps\_data()

```
void collect_gps_data ()
```

Definition at line 29 of file [gps\\_collector.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.45.3 Variable Documentation

#### 8.45.3.1 pause\_gps\_collection

```
volatile bool pause_gps_collection [extern]
```

Definition at line 8 of file [main.cpp](#).

#### 8.45.3.2 nmea\_data

```
NMEAData nmea_data [extern]
```

Definition at line 3 of file [NMEA\\_data.cpp](#).

## 8.46 gps\_collector.cpp

[Go to the documentation of this file.](#)

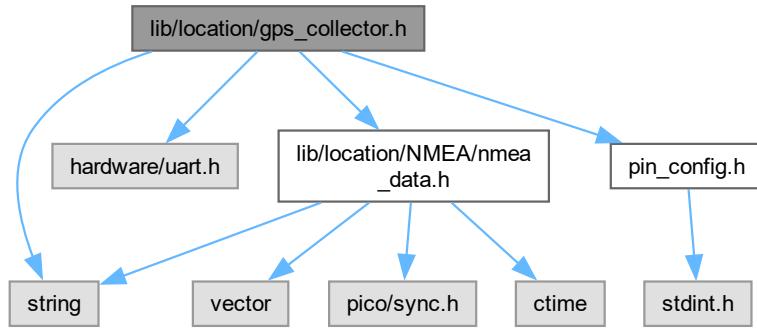
```

00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/gps_collector.cpp
00002 #include "lib/location/gps_collector.h"
00003 #include "utils.h"
00004 #include "pico/time.h"
00005 #include "lib/location/NMEA/nmea_data.h"
00006 #include "event_manager.h"
00007 #include <vector>
00008 #include <ctime>
00009 #include <cstring>
00010 #include "DS3231.h"
00011 #include <sstream>
00012
00013 extern volatile bool pause_gps_collection;
00014
00015 #define MAX_RAW_DATA_LENGTH 256
00016
00017 extern NMEAData nmea_data;
00018
00019 std::vector<std::string> splitString(const std::string& str, char delimiter) {
00020     std::vector<std::string> tokens;
00021     std::stringstream ss(str);
00022     std::string token;
00023     while (std::getline(ss, token, delimiter)) {
00024         tokens.push_back(token);
00025     }
00026     return tokens;
00027 }
00028
00029 void collect_gps_data() {
00030
00031     if (pause_gps_collection) {
00032         return;
00033     }
00034
00035     static char raw_data_buffer[MAX_RAW_DATA_LENGTH];
00036     static int raw_data_index = 0;
00037
00038     while (uart_is_readable(GPS_UART_PORT)) {
00039         char c = uart_getc(GPS_UART_PORT);
00040
00041         if (c == '\r' || c == '\n') {
00042             // End of message
00043             if (raw_data_index > 0) {
00044                 raw_data_buffer[raw_data_index] = '\0';
00045                 std::string message(raw_data_buffer);
00046                 raw_data_index = 0;
00047
00048                 // Split the message into tokens
00049                 std::vector<std::string> tokens = splitString(message, ',');
00050
00051                 // Update the global vectors based on the sentence type
00052                 if (message.find("$GPRMC") == 0) {
00053                     nmea_data.update_rmc_tokens(tokens);
00054                 } else if (message.find("$GPGGA") == 0) {
00055                     nmea_data.update_gga_tokens(tokens);
00056                 }
00057             }
00058         } else {
00059             // Append to buffer
00060             if (raw_data_index < MAX_RAW_DATA_LENGTH - 1) {
00061                 raw_data_buffer[raw_data_index++] = c;
00062             } else {
00063                 raw_data_index = 0;
00064             }
00065         }
00066     }
00067 }
```

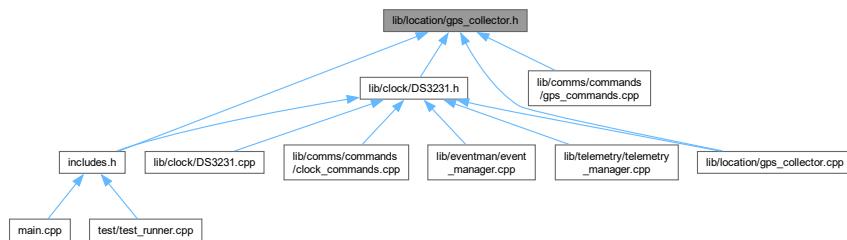
## 8.47 lib/location/gps\_collector.h File Reference

```
#include <string>
#include "hardware/uart.h"
#include "lib/location/NMEA/nmea_data.h"
```

```
#include "pin_config.h"
Include dependency graph for gps_collector.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- void [collect\\_gps\\_data \(\)](#)

### 8.47.1 Function Documentation

#### 8.47.1.1 [collect\\_gps\\_data\(\)](#)

```
void collect_gps_data ()
```

Definition at line 29 of file [gps\\_collector.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



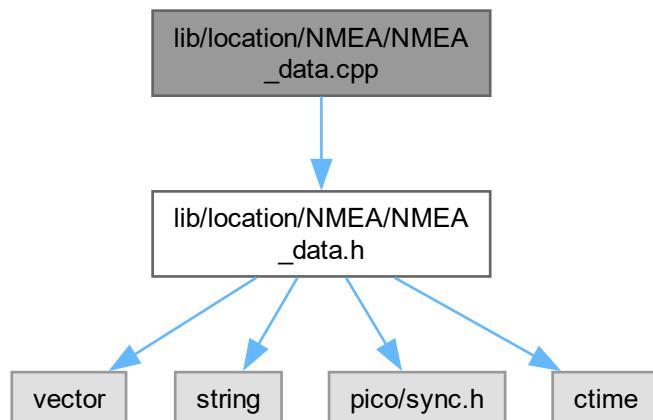
## 8.48 gps\_collector.h

[Go to the documentation of this file.](#)

```
00001 #ifndef GPS_COLLECTOR_H
00002 #define GPS_COLLECTOR_H
00003
00004 #include <string>
00005 #include "hardware/uart.h"
00006 #include "lib/location/NMEA/nmea_data.h" // Include the new header
00007 #include "pin_config.h"
00008
00009 // Function to collect GPS data from the UART
00010 void collect_gps_data();
00011
00012 #endif
```

## 8.49 lib/location/NMEA/NMEA\_data.cpp File Reference

```
#include "lib/location/NMEA/NMEA_data.h"
Include dependency graph for NMEA_data.cpp:
```



## Variables

- `NMEAData nmea_data`

### 8.49.1 Variable Documentation

#### 8.49.1.1 `nmea_data`

`NMEAData nmea_data`

Definition at line 3 of file `NMEA_data.cpp`.

## 8.50 `NMEA_data.cpp`

[Go to the documentation of this file.](#)

```
00001 #include "lib/location/NMEA/NMEA_data.h"
00002
00003 NMEAData nmea_data;
00004
00005 NMEAData::NMEAData() {
00006     mutex_init(&rmc_mutex_);
00007     mutex_init(&gga_mutex_);
00008 }
00009
0010 void NMEAData::update_rmc_tokens(const std::vector<std::string>& tokens) {
0011     mutex_enter_blocking(&rmc_mutex_);
0012     rmc_tokens_ = tokens;
0013     mutex_exit(&rmc_mutex_);
0014 }
0015
0016 void NMEAData::update_gga_tokens(const std::vector<std::string>& tokens) {
0017     mutex_enter_blocking(&gga_mutex_);
0018     gga_tokens_ = tokens;
0019     mutex_exit(&gga_mutex_);
0020 }
0021
0022 std::vector<std::string> NMEAData::get_rmc_tokens() const {
0023     mutex_enter_blocking(const_cast<mutex_t*>(&rmc_mutex_));
0024     std::vector<std::string> copy = rmc_tokens_;
0025     mutex_exit(const_cast<mutex_t*>(&rmc_mutex_));
0026     return copy;
0027 }
0028
0029 std::vector<std::string> NMEAData::get_gga_tokens() const {
0030     mutex_enter_blocking(const_cast<mutex_t*>(&gga_mutex_));
0031     std::vector<std::string> copy = gga_tokens_;
0032     mutex_exit(const_cast<mutex_t*>(&gga_mutex_));
0033     return copy;
0034 }
0035
0036 bool NMEAData::has_valid_time() const {
0037     return rmc_tokens_.size() >= 10 && rmc_tokens_[1].length() > 5;
0038 }
0039
0040 time_t NMEAData::get_unix_time() const {
0041     if (!has_valid_time()) {
0042         return 0; // Invalid time
0043     }
0044
0045     // Parse date and time from RMC tokens
0046     // Format: hhmmss.sss,A,ddmm.mmmm,N,dddmmyy,W,speed,course,ddmmyy
0047     std::string time_str = rmc_tokens_[1]; // hhmmss.sss
0048     std::string date_str = rmc_tokens_[9]; // ddmmyy
0049
0050     if (time_str.length() < 6 || date_str.length() < 6) {
0051         return 0;
0052     }
0053
0054     struct tm timeinfo = {0};
0055
0056     // Parse time: hours (0-1), minutes (2-3), seconds (4-5)
0057     timeinfo.tm_hour = std::stoi(time_str.substr(0, 2));
```

```

00058     timeinfo.tm_min = std::stoi(time_str.substr(2, 2));
00059     timeinfo.tm_sec = std::stoi(time_str.substr(4, 2));
00060
00061     // Parse date: day (0-1), month (2-3), year (4-5)
00062     timeinfo.tm_mday = std::stoi(date_str.substr(0, 2));
00063     timeinfo.tm_mon = std::stoi(date_str.substr(2, 2)) - 1; // Months from 0-11
00064     timeinfo.tm_year = std::stoi(date_str.substr(4, 2)) + 100; // Years since 1900
00065
00066     // Convert to unix time
00067     return mktime(&timeinfo);
00068 }

```

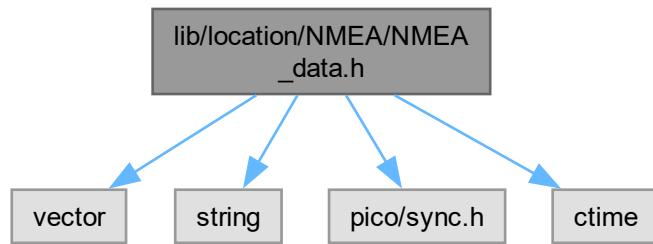
## 8.51 lib/location/NMEA/NMEA\_data.h File Reference

```

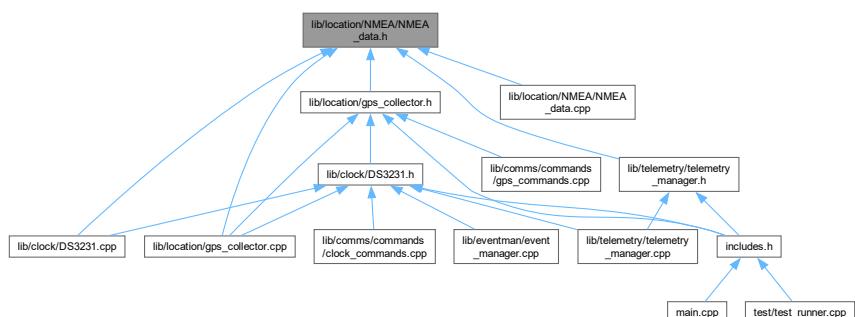
#include <vector>
#include <string>
#include "pico/sync.h"
#include <ctime>

```

Include dependency graph for NMEA\_data.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [NMEAData](#)

## Variables

- NMEAData nmea\_data

### 8.51.1 Variable Documentation

#### 8.51.1.1 nmea\_data

NMEAData nmea\_data [extern]

Definition at line 3 of file [NMEA\\_data.cpp](#).

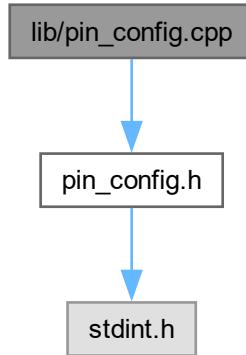
## 8.52 NMEA\_data.h

[Go to the documentation of this file.](#)

```
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/nmea_data.h
00002 #ifndef NMEA_DATA_H
00003 #define NMEA_DATA_H
00004
00005 #include <vector>
00006 #include <string>
00007 #include "pico/sync.h"
00008 #include <ctime>
00009
00010 class NMEAData {
00011 public:
00012     NMEAData();
00013     void update_rmc_tokens(const std::vector<std::string>& tokens);
00014     void update_gga_tokens(const std::vector<std::string>& tokens);
00015
00016     std::vector<std::string> get_rmc_tokens() const;
00017     std::vector<std::string> get_gga_tokens() const;
00018
00019     bool has_valid_time() const;
00020
00021     time_t get_unix_time() const;
00022
00023 private:
00024     std::vector<std::string> rmc_tokens_;
00025     std::vector<std::string> gga_tokens_;
00026     mutex_t rmc_mutex_;
00027     mutex_t gga_mutex_;
00028 };
00029
00030 extern NMEAData nmea_data;
00031
00032 #endif
```

## 8.53 lib/pin\_config.cpp File Reference

```
#include "pin_config.h"  
Include dependency graph for pin_config.cpp:
```



### Variables

- const int `lora_cs_pin` = 17
- const int `lora_reset_pin` = 22
- const int `lora_irq_pin` = 28
- uint8\_t `lora_address_local` = 37
- uint8\_t `lora_address_remote` = 21

### 8.53.1 Variable Documentation

#### 8.53.1.1 lora\_cs\_pin

```
const int lora_cs_pin = 17
```

Definition at line 4 of file [pin\\_config.cpp](#).

#### 8.53.1.2 lora\_reset\_pin

```
const int lora_reset_pin = 22
```

Definition at line 5 of file [pin\\_config.cpp](#).

#### 8.53.1.3 lora\_irq\_pin

```
const int lora_irq_pin = 28
```

Definition at line 6 of file [pin\\_config.cpp](#).

#### 8.53.1.4 lora\_address\_local

```
uint8_t lora_address_local = 37
```

Definition at line 8 of file [pin\\_config.cpp](#).

#### 8.53.1.5 lora\_address\_remote

```
uint8_t lora_address_remote = 21
```

Definition at line 9 of file [pin\\_config.cpp](#).

### 8.54 pin\_config.cpp

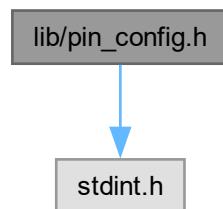
[Go to the documentation of this file.](#)

```
00001 #include "pin_config.h"
00002
00003 // LoRa constants
00004 const int lora_cs_pin = 17;           // LoRa radio chip select
00005 const int lora_reset_pin = 22;        // LoRa radio reset
00006 const int lora_irq_pin = 28;          // LoRa hardware interrupt pin
00007
00008 uint8_t lora_address_local = 37;      // address of this device
00009 uint8_t lora_address_remote = 21;
```

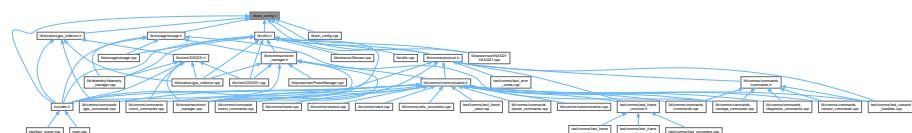
### 8.55 lib/pin\_config.h File Reference

```
#include <stdint.h>
```

Include dependency graph for pin\_config.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define DEBUG_UART_PORT uart0`
- `#define DEBUG_UART_BAUD_RATE 115200`
- `#define DEBUG_UART_TX_PIN 0`
- `#define DEBUG_UART_RX_PIN 1`
- `#define MAIN_I2C_PORT i2c1`
- `#define MAIN_I2C_SDA_PIN 6`
- `#define MAIN_I2C_SCL_PIN 7`
- `#define GPS_UART_PORT uart1`
- `#define GPS_UART_BAUD_RATE 9600`
- `#define GPS_UART_TX_PIN 8`
- `#define GPS_UART_RX_PIN 9`
- `#define GPS_POWER_ENABLE_PIN 14`
- `#define BUFFER_SIZE 85`
- `#define SD_SPI_PORT spi1`
- `#define SD_MISO_PIN 12`
- `#define SD_MOSI_PIN 11`
- `#define SD_SCK_PIN 10`
- `#define SD_CS_PIN 13`
- `#define SD_CARD_DETECT_PIN 28`
- `#define SX1278_MISO 16`
- `#define SX1278_CS 17`
- `#define SX1278_SCK 18`
- `#define SX1278_MOSI 19`
- `#define SPI_PORT spi0`
- `#define READ_BIT 0x80`
- `#define LORA_DEFAULT_SPI spi0`
- `#define LORA_DEFAULT_SPI_FREQUENCY 8E6`
- `#define LORA_DEFAULT_SS_PIN 17`
- `#define LORA_DEFAULT_RESET_PIN 22`
- `#define LORA_DEFAULT_DIO0_PIN 20`
- `#define PA_OUTPUT_RFO_PIN 11`
- `#define PA_OUTPUT_PA_BOOST_PIN 12`

## Variables

- `const int lora_cs_pin`
- `const int lora_reset_pin`
- `const int lora_irq_pin`
- `uint8_t lora_address_local`
- `uint8_t lora_address_remote`

### 8.55.1 Macro Definition Documentation

#### 8.55.1.1 DEBUG\_UART\_PORT

```
#define DEBUG_UART_PORT uart0
```

Definition at line 8 of file [pin\\_config.h](#).

### 8.55.1.2 DEBUG\_UART\_BAUD\_RATE

```
#define DEBUG_UART_BAUD_RATE 115200
```

Definition at line [9](#) of file [pin\\_config.h](#).

### 8.55.1.3 DEBUG\_UART\_TX\_PIN

```
#define DEBUG_UART_TX_PIN 0
```

Definition at line [11](#) of file [pin\\_config.h](#).

### 8.55.1.4 DEBUG\_UART\_RX\_PIN

```
#define DEBUG_UART_RX_PIN 1
```

Definition at line [12](#) of file [pin\\_config.h](#).

### 8.55.1.5 MAIN\_I2C\_PORT

```
#define MAIN_I2C_PORT i2c1
```

Definition at line [14](#) of file [pin\\_config.h](#).

### 8.55.1.6 MAIN\_I2C\_SDA\_PIN

```
#define MAIN_I2C_SDA_PIN 6
```

Definition at line [15](#) of file [pin\\_config.h](#).

### 8.55.1.7 MAIN\_I2C\_SCL\_PIN

```
#define MAIN_I2C_SCL_PIN 7
```

Definition at line [16](#) of file [pin\\_config.h](#).

### 8.55.1.8 GPS\_UART\_PORT

```
#define GPS_UART_PORT uart1
```

Definition at line [19](#) of file [pin\\_config.h](#).

### 8.55.1.9 GPS\_UART\_BAUD\_RATE

```
#define GPS_UART_BAUD_RATE 9600
```

Definition at line [20](#) of file [pin\\_config.h](#).

### 8.55.1.10 GPS\_UART\_TX\_PIN

```
#define GPS_UART_TX_PIN 8
```

Definition at line 21 of file [pin\\_config.h](#).

### 8.55.1.11 GPS\_UART\_RX\_PIN

```
#define GPS_UART_RX_PIN 9
```

Definition at line 22 of file [pin\\_config.h](#).

### 8.55.1.12 GPS\_POWER\_ENABLE\_PIN

```
#define GPS_POWER_ENABLE_PIN 14
```

Definition at line 23 of file [pin\\_config.h](#).

### 8.55.1.13 BUFFER\_SIZE

```
#define BUFFER_SIZE 85
```

Definition at line 25 of file [pin\\_config.h](#).

### 8.55.1.14 SD\_SPI\_PORT

```
#define SD_SPI_PORT spil
```

Definition at line 28 of file [pin\\_config.h](#).

### 8.55.1.15 SD\_MISO\_PIN

```
#define SD_MISO_PIN 12
```

Definition at line 29 of file [pin\\_config.h](#).

### 8.55.1.16 SD\_MOSI\_PIN

```
#define SD_MOSI_PIN 11
```

Definition at line 30 of file [pin\\_config.h](#).

### 8.55.1.17 SD\_SCK\_PIN

```
#define SD_SCK_PIN 10
```

Definition at line 31 of file [pin\\_config.h](#).

### 8.55.1.18 SD\_CS\_PIN

```
#define SD_CS_PIN 13
```

Definition at line 32 of file [pin\\_config.h](#).

### 8.55.1.19 SD\_CARD\_DETECT\_PIN

```
#define SD_CARD_DETECT_PIN 28
```

Definition at line 33 of file [pin\\_config.h](#).

### 8.55.1.20 SX1278\_MISO

```
#define SX1278_MISO 16
```

Definition at line 35 of file [pin\\_config.h](#).

### 8.55.1.21 SX1278\_CS

```
#define SX1278_CS 17
```

Definition at line 36 of file [pin\\_config.h](#).

### 8.55.1.22 SX1278\_SCK

```
#define SX1278_SCK 18
```

Definition at line 37 of file [pin\\_config.h](#).

### 8.55.1.23 SX1278\_MOSI

```
#define SX1278_MOSI 19
```

Definition at line 38 of file [pin\\_config.h](#).

### 8.55.1.24 SPI\_PORT

```
#define SPI_PORT spi0
```

Definition at line 40 of file [pin\\_config.h](#).

### 8.55.1.25 READ\_BIT

```
#define READ_BIT 0x80
```

Definition at line 41 of file [pin\\_config.h](#).

### 8.55.1.26 LORA\_DEFAULT\_SPI

```
#define LORA_DEFAULT_SPI spio
```

Definition at line 43 of file [pin\\_config.h](#).

### 8.55.1.27 LORA\_DEFAULT\_SPI\_FREQUENCY

```
#define LORA_DEFAULT_SPI_FREQUENCY 8E6
```

Definition at line 44 of file [pin\\_config.h](#).

### 8.55.1.28 LORA\_DEFAULT\_SS\_PIN

```
#define LORA_DEFAULT_SS_PIN 17
```

Definition at line 45 of file [pin\\_config.h](#).

### 8.55.1.29 LORA\_DEFAULT\_RESET\_PIN

```
#define LORA_DEFAULT_RESET_PIN 22
```

Definition at line 46 of file [pin\\_config.h](#).

### 8.55.1.30 LORA\_DEFAULT\_DIO0\_PIN

```
#define LORA_DEFAULT_DIO0_PIN 20
```

Definition at line 47 of file [pin\\_config.h](#).

### 8.55.1.31 PA\_OUTPUT\_RFO\_PIN

```
#define PA_OUTPUT_RFO_PIN 11
```

Definition at line 49 of file [pin\\_config.h](#).

### 8.55.1.32 PA\_OUTPUT\_PA\_BOOST\_PIN

```
#define PA_OUTPUT_PA_BOOST_PIN 12
```

Definition at line 50 of file [pin\\_config.h](#).

## 8.55.2 Variable Documentation

### 8.55.2.1 lora\_cs\_pin

```
const int lora_cs_pin [extern]
```

Definition at line 4 of file [pin\\_config.cpp](#).

### 8.55.2.2 lora\_reset\_pin

```
const int lora_reset_pin [extern]
```

Definition at line 5 of file [pin\\_config.cpp](#).

### 8.55.2.3 lora\_irq\_pin

```
const int lora_irq_pin [extern]
```

Definition at line 6 of file [pin\\_config.cpp](#).

### 8.55.2.4 lora\_address\_local

```
uint8_t lora_address_local [extern]
```

Definition at line 8 of file [pin\\_config.cpp](#).

### 8.55.2.5 lora\_address\_remote

```
uint8_t lora_address_remote [extern]
```

Definition at line 9 of file [pin\\_config.cpp](#).

## 8.56 pin\_config.h

[Go to the documentation of this file.](#)

```

00001 // pin_config.h
00002 #include <stdint.h>
00003
00004 #ifndef PIN_CONFIG_H
00005 #define PIN_CONFIG_H
00006
00007 //DEBUG uart
00008 #define DEBUG_UART_PORT uart0
00009 #define DEBUG_UART_BAUD_RATE 115200
00010
00011 #define DEBUG_UART_TX_PIN 0
00012 #define DEBUG_UART_RX_PIN 1
00013
00014 #define MAIN_I2C_PORT i2c1
00015 #define MAIN_I2C_SDA_PIN 6
00016 #define MAIN_I2C_SCL_PIN 7
00017
00018 // GPS configuration
00019 #define GPS_UART_PORT uart1
00020 #define GPS_UART_BAUD_RATE 9600
00021 #define GPS_UART_TX_PIN 8
00022 #define GPS_UART_RX_PIN 9
00023 #define GPS_POWER_ENABLE_PIN 14
00024
00025 #define BUFFER_SIZE 85 // NMEA sentences are usually under 85 chars
00026
00027 // SPI configuration for SD card
00028 #define SD_SPI_PORT spil
00029 #define SD_MISO_PIN 12
00030 #define SD_MOSI_PIN 11
00031 #define SD_SCK_PIN 10
00032 #define SD_CS_PIN 13
00033 #define SD_CARD_DETECT_PIN 28
00034
00035 #define SX1278_MISO 16
00036 #define SX1278_CS 17
00037 #define SX1278_SCK 18
00038 #define SX1278_MOSI 19
00039
00040 #define SPI_PORT spi0
00041 #define READ_BIT 0x80
00042
00043 #define LORA_DEFAULT_SPI spi0
00044 #define LORA_DEFAULT_SPI_FREQUENCY 8E6
00045 #define LORA_DEFAULT_SS_PIN 17
00046 #define LORA_DEFAULT_RESET_PIN 22
00047 #define LORA_DEFAULT_DIO0_PIN 20
00048
00049 #define PA_OUTPUT_RFO_PIN 11
00050 #define PA_OUTPUT_PA_BOOST_PIN 12
00051
00052
00053
00054 // LoRa constants - declare as extern
00055 extern const int lora_cs_pin; // LoRa radio chip select
00056 extern const int lora_reset_pin; // LoRa radio reset
00057 extern const int lora_irq_pin; // LoRa hardware interrupt pin
00058 extern uint8_t lora_address_local; // address of this device
00059 extern uint8_t lora_address_remote; // destination to send to
00060
00061
00062 #endif // PIN_CONFIG_H

```

## 8.57 lib/powerman/INA3221/INA3221.cpp File Reference

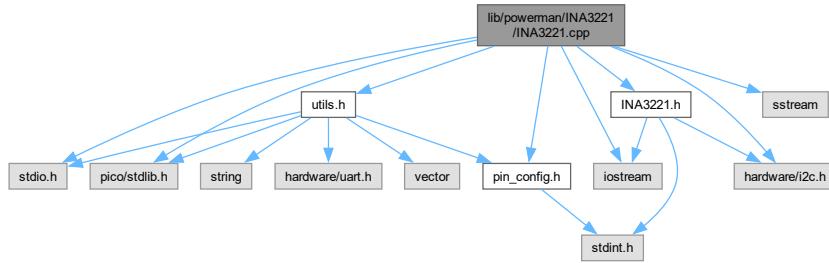
Implementation of the [INA3221](#) power monitor driver.

```

#include "INA3221.h"
#include <stdio.h>
#include "pico/stl.h"
#include "hardware/i2c.h"
#include <iostream>

```

```
#include "pin_config.h"
#include "utils.h"
#include <sstream>
Include dependency graph for INA3221.cpp:
```



### 8.57.1 Detailed Description

Implementation of the [INA3221](#) power monitor driver.

This file contains the implementation for the [INA3221](#) triple-channel power monitor, providing functionality for voltage, current, and power monitoring with alert capabilities.

Definition in file [INA3221.cpp](#).

## 8.58 INA3221.cpp

[Go to the documentation of this file.](#)

```

00001 #include "INA3221.h"
00002 #include <stdio.h>
00003 #include "pico/stlplib.h"
00004 #include "hardware/i2c.h"
00005 #include <iostream>
00006 #include "pin_config.h"
00007 #include "utils.h"
00008 #include <sstream>
00009
00010
00017
00018
00038
00039
00046 INA3221::INA3221(in3221_addr_t addr, i2c_inst_t* i2c)
00047     : _i2c_addr(addr), _i2c(i2c) {}
00048
00049
00056 bool INA3221::begin() {
00057     uart_print("INA3221 initializing...", VerboseLevel::DEBUG);
00058
00059     _shuntRes[0] = 10;
00060     _shuntRes[1] = 10;
00061     _shuntRes[2] = 10;
00062
00063     _filterRes[0] = 10;
00064     _filterRes[1] = 10;
00065     _filterRes[2] = 10;
00066
00067     uint16_t manuf_id = get_manufacturer_id();
00068     uint16_t die_id = get_die_id();
00069     std::stringstream ss;
00070     ss << "INA3221 Manufacturer ID: 0x" << std::hex << manuf_id
00071             << ", Die ID: 0x" << die_id;
00072     uart_print(ss.str(), VerboseLevel::INFO);
00073 }
```

```
00073     if (manuf_id == 0x5449 && die_id == 0x3220) {
00074         uart_print("INA3221 found and initialized.", VerbosityLevel::DEBUG);
00075         return true;
00076     } else {
00077         uart_print("INA3221 initialization failed. Incorrect IDs.", VerbosityLevel::ERROR);
00078         return false;
00079     }
00080 }
00081
00082 }
00083
00084
00085 void INA3221::reset(){
00086     conf_reg_t conf_reg;
00087
00088     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00089     conf_reg.reset = 1;
00090     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00091 }
00092
00093
00094 uint16_t INA3221::get_manufacturer_id() {
00095     uint16_t id = 0;
00096     _read(INA3221_REG_MANUF_ID, &id);
00097     return id;
00098 }
00099
00100
00101 uint16_t INA3221::get_die_id() {
00102     uint16_t id = 0;
00103     _read(INA3221_REG_DIE_ID, &id);
00104     return id;
00105 }
00106
00107
00108 }
00109
00110
00111
00112
00113 uint16_t INA3221::read_register(ina3221_reg_t reg){
00114     uint16_t val = 0;
00115     _read(reg, &val);
00116     return val;
00117 }
00118
00119
00120 }
00121
00122
00123
00124 //configure
00125
00126
00127 void INA3221::set_mode_power_down(){
00128     conf_reg_t conf_reg;
00129
00130     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00131     conf_reg.mode_bus_en = 0;
00132     conf_reg.mode_continious_en =0 ;
00133     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00134 }
00135
00136
00137
00138 void INA3221::set_mode_continuous(){
00139     conf_reg_t conf_reg;
00140
00141     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00142     conf_reg.mode_continious_en =1;
00143     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00144 }
00145
00146
00147
00148 void INA3221::set_mode_triggered(){
00149     conf_reg_t conf_reg;
00150
00151     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00152     conf_reg.mode_continious_en = 0;
00153     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00154 }
00155
00156
00157
00158 void INA3221::set_shunt_measurement_enable(){
00159     conf_reg_t conf_reg;
00160
00161     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00162     conf_reg.mode_shunt_en = 1;
00163     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00164 }
00165
00166
00167
00168 void INA3221::set_shunt_measurement_disable(){
00169     conf_reg_t conf_reg;
00170
00171     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00172     conf_reg.mode_shunt_en = 0;
00173     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00174 }
```

```

00204 }
00205
00206
00211 void INA3221::set_bus_measurement_enable() {
00212     conf_reg_t conf_reg;
00213
00214     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00215     conf_reg.mode_bus_en = 1;
00216     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00217 }
00218
00219
00224 void INA3221::set_bus_measurement_disable() {
00225     conf_reg_t conf_reg;
00226
00227     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00228     conf_reg.mode_bus_en = 0;
00229     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00230 }
00231
00232
00238 void INA3221::set_averaging_mode(ina3221_avg_mode_t mode) {
00239     conf_reg_t conf_reg;
00240
00241     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00242     conf_reg.avg_mode = mode;
00243     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00244 }
00245
00246
00252 void INA3221::set_bus_conversion_time(ina3221_conv_time_t convTime) {
00253     conf_reg_t conf_reg;
00254
00255     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00256     conf_reg.bus_conv_time = convTime;
00257     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00258 }
00259
00260
00266 void INA3221::set_shunt_conversion_time(ina3221_conv_time_t convTime) {
00267     conf_reg_t conf_reg;
00268
00269     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00270     conf_reg.shunt_conv_time = convTime;
00271     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00272 }
00273
00274
00275 //get measurement
00282 int32_t INA3221::get_shunt_voltage(ina3221_ch_t channel) {
00283     int32_t res;
00284     ina3221_reg_t reg;
00285     uint16_t val_raw = 0;
00286
00287     switch(channel){
00288         case INA3221_CH1:
00289             reg = INA3221_REG_CH1_SHUNTV;
00290             break;
00291         case INA3221_CH2:
00292             reg = INA3221_REG_CH2_SHUNTV;
00293             break;
00294         case INA3221_CH3:
00295             reg = INA3221_REG_CH3_SHUNTV;
00296             break;
00297     }
00298
00299     _read(reg, &val_raw);
00300
00301     res = (int16_t) (val_raw >> 3);
00302     res *= SHUNT_VOLTAGE_LSB_UV;
00303
00304     return res;
00305 }
00306
00307
00314 float INA3221::get_current_ma(ina3221_ch_t channel) {
00315     int32_t shunt_uV = 0;
00316     float current_A = 0;
00317
00318     shunt_uV = get_shunt_voltage(channel);
00319     current_A = shunt_uV / (int32_t)_shuntRes[channel] / 1000.0;;
00320     return current_A;
00321 }
00322
00323
00330 float INA3221::get_voltage(ina3221_ch_t channel) {
00331     float voltage_V = 0.0;

```

```

00332     ina3221_reg_t reg;
00333     uint16_t val_raw = 0;
00334
00335     switch(channel){
00336         case INA3221_CH1:
00337             reg = INA3221_REG_CH1_BUSV;
00338             break;
00339         case INA3221_CH2:
00340             reg = INA3221_REG_CH2_BUSV;
00341             break;
00342         case INA3221_CH3:
00343             reg = INA3221_REG_CH3_BUSV;
00344             break;
00345     }
00346
00347     _read(reg, &val_raw);
00348     voltage_V = val_raw / 1000.0;
00349     return voltage_V;
00350 }
00351
00352
00353 // alerts
00360 void INA3221::set_warn_alert_limit(ina3221_ch_t channel, float voltage_v) {
00361     ina3221_reg_t reg;
00362     uint16_t val = (uint16_t)(voltage_v * 1000); // Convert V to mV
00363
00364     switch(channel) {
00365         case INA3221_CH1:
00366             reg = INA3221_REG_CH1_WARNING_ALERT_LIM;
00367             break;
00368         case INA3221_CH2:
00369             reg = INA3221_REG_CH2_WARNING_ALERT_LIM;
00370             break;
00371         case INA3221_CH3:
00372             reg = INA3221_REG_CH3_WARNING_ALERT_LIM;
00373             break;
00374     }
00375     _write(reg, &val);
00376 }
00377
00378
00385 void INA3221::set_crit_alert_limit(ina3221_ch_t channel, float voltage_v) {
00386     ina3221_reg_t reg;
00387     uint16_t val = (uint16_t)(voltage_v * 1000); // Convert V to mV
00388
00389     switch(channel) {
00390         case INA3221_CH1:
00391             reg = INA3221_REG_CH1_CRIT_ALERT_LIM;
00392             break;
00393         case INA3221_CH2:
00394             reg = INA3221_REG_CH2_CRIT_ALERT_LIM;
00395             break;
00396         case INA3221_CH3:
00397             reg = INA3221_REG_CH3_CRIT_ALERT_LIM;
00398             break;
00399     }
00400     _write(reg, &val);
00401 }
00402
00403
00410 void INA3221::set_power_valid_limit(float voltage_upper_v, float voltage_lower_v) {
00411     uint16_t val;
00412
00413     val = (uint16_t)(voltage_upper_v * 1000);
00414     _write(INA3221_REG_PWR_VALID_HI_LIM, &val);
00415
00416     val = (uint16_t)(voltage_lower_v * 1000);
00417     _write(INA3221_REG_PWR_VALID_LO_LIM, &val);
00418 }
00419
00420
00426 void INA3221::enable_alerts() {
00427     masken_reg_t masken;
00428     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00429
00430     masken.warn_alert_ch1 = 1;
00431     masken.warn_alert_ch2 = 1;
00432     masken.warn_alert_ch3 = 1;
00433     masken.crit_alert_ch1 = 1;
00434     masken.crit_alert_ch2 = 1;
00435     masken.crit_alert_ch3 = 1;
00436     masken.pwr_valid_alert = 1;
00437
00438     _write(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00439 }
00440
00441

```

```

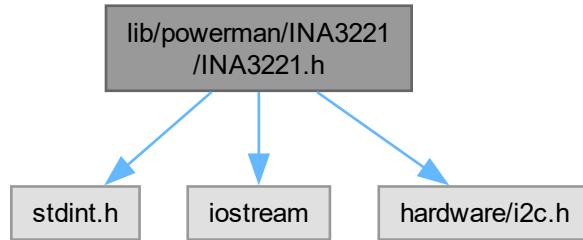
00448 bool INA3221::get_warn_alert(ina3221_ch_t channel) {
00449     masken_reg_t masken;
00450     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00451
00452     switch(channel) {
00453         case INA3221_CH1: return masken.warn_alert_ch1;
00454         case INA3221_CH2: return masken.warn_alert_ch2;
00455         case INA3221_CH3: return masken.warn_alert_ch3;
00456         default: return false;
00457     }
00458 }
00459
00460
00461 bool INA3221::get_crit_alert(ina3221_ch_t channel) {
00462     masken_reg_t masken;
00463     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00464
00465     switch(channel) {
00466         case INA3221_CH1: return masken.crit_alert_ch1;
00467         case INA3221_CH2: return masken.crit_alert_ch2;
00468         case INA3221_CH3: return masken.crit_alert_ch3;
00469         default: return false;
00470     }
00471 }
00472
00473
00474
00475 bool INA3221::get_power_valid_alert() {
00476     masken_reg_t masken;
00477     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00478     return masken.pwr_valid_alert;
00479 }
00480
00481
00482
00483 void INA3221::set_alert_latch(bool enable) {
00484     masken_reg_t masken;
00485     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00486     masken.warn_alert_latch_en = enable;
00487     masken.crit_alert_latch_en = enable;
00488     _write(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00489 }
00490
00491
00492
00493 // private
00494 void INA3221::_read(ina3221_reg_t reg, uint16_t *val) {
00495     uint8_t reg_buf = reg;
00496     uint8_t data[2];
00497
00498     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, &reg_buf, 1, true);
00499     if (ret != 1) {
00500         std::cerr << "Failed to write register address to I2C device." << std::endl;
00501         return;
00502     }
00503
00504
00505
00506     ret = i2c_read_blocking(MAIN_I2C_PORT, _i2c_addr, data, 2, false);
00507     if (ret != 2) {
00508         std::cerr << "Failed to read data from I2C device." << std::endl;
00509         return;
00510     }
00511
00512
00513     *val = (data[0] << 8) | data[1];
00514 }
00515
00516
00517
00518 void INA3221::_write(ina3221_reg_t reg, uint16_t *val) {
00519     uint8_t buf[3];
00520     buf[0] = reg;
00521     buf[1] = (*val >> 8) & 0xFF; // MSB
00522     buf[2] = (*val) & 0xFF; // LSB
00523
00524     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, buf, 3, false);
00525     if (ret != 3) {
00526         std::cerr << "Failed to write data to I2C device." << std::endl;
00527     }
00528 }
00529
00530 }
```

## 8.59 lib/powerman/INA3221/INA3221.h File Reference

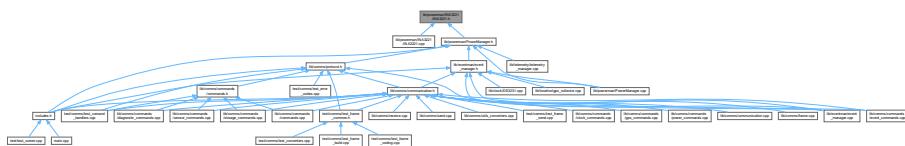
Header file for the [INA3221](#) triple-channel power monitor driver.

```
#include <stdint.h>
#include <iostream>
```

```
#include <hardware/i2c.h>
Include dependency graph for INA3221.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [INA3221](#)  
*INA3221 Triple-Channel Power Monitor driver class.*
- struct [INA3221::conf\\_reg\\_t](#)  
*Configuration register bit fields.*
- struct [INA3221::masken\\_reg\\_t](#)  
*Mask/Enable register bit fields.*

## Enumerations

- enum [ina3221\\_addr\\_t](#) { [INA3221\\_ADDR40\\_GND](#) = 0b1000000 , [INA3221\\_ADDR41\\_VCC](#) = 0b1000001 , [INA3221\\_ADDR42\\_SDA](#) = 0b1000010 , [INA3221\\_ADDR43\\_SCL](#) = 0b1000011 }
- enum [ina3221\\_ch\\_t](#) { [INA3221\\_CH1](#) = 0 , [INA3221\\_CH2](#) , [INA3221\\_CH3](#) }
- enum [ina3221\\_reg\\_t](#) {  
[INA3221\\_REG\\_CONF](#) = 0 , [INA3221\\_REG\\_CH1\\_SHUNTV](#) , [INA3221\\_REG\\_CH1\\_BUSV](#) , [INA3221\\_REG\\_CH2\\_SHUNTV](#) ,  
[INA3221\\_REG\\_CH2\\_BUSV](#) , [INA3221\\_REG\\_CH3\\_SHUNTV](#) , [INA3221\\_REG\\_CH3\\_BUSV](#) , [INA3221\\_REG\\_CH1\\_CRIT\\_ALEF](#) ,  
[INA3221\\_REG\\_CH1\\_WARNING\\_ALERT\\_LIM](#) , [INA3221\\_REG\\_CH2\\_CRIT\\_ALERT\\_LIM](#) , [INA3221\\_REG\\_CH2\\_WARNING\\_A](#) ,  
[INA3221\\_REG\\_CH3\\_CRIT\\_ALERT\\_LIM](#) , [INA3221\\_REG\\_SHUNTV\\_SUM](#) , [INA3221\\_REG\\_SHUNTV\\_SUM\\_LIM](#) ,  
[INA3221\\_REG\\_MASK\\_ENABLE](#) , [INA3221\\_REG\\_PWR\\_VALID\\_HI\\_LIM](#) , [INA3221\\_REG\\_PWR\\_VALID\\_LO\\_LIM](#) , [INA3221\\_REG\\_MANUF\\_ID](#) = 0xFE , [INA3221\\_REG\\_DIE\\_ID](#) = 0xFF }

*Register addresses for [INA3221](#).*

- enum `ina3221_conv_time_t` {
   
INA3221\_REG\_CONF\_CT\_140US = 0 , INA3221\_REG\_CONF\_CT\_204US , INA3221\_REG\_CONF\_CT\_332US  
, INA3221\_REG\_CONF\_CT\_588US ,  
INA3221\_REG\_CONF\_CT\_1100US , INA3221\_REG\_CONF\_CT\_2116US , INA3221\_REG\_CONF\_CT\_4156US  
, INA3221\_REG\_CONF\_CT\_8244US }
- Conversion time settings.*
- enum `ina3221_avg_mode_t` {
   
INA3221\_REG\_CONF\_AVG\_1 = 0 , INA3221\_REG\_CONF\_AVG\_4 , INA3221\_REG\_CONF\_AVG\_16 ,  
INA3221\_REG\_CONF\_AVG\_64 ,  
INA3221\_REG\_CONF\_AVG\_128 , INA3221\_REG\_CONF\_AVG\_256 , INA3221\_REG\_CONF\_AVG\_512 ,  
INA3221\_REG\_CONF\_AVG\_1024 }

*Averaging mode settings.*

## Variables

- const int `INA3221_CH_NUM` = 3
   
*Number of channels in [INA3221](#).*
- const int `SHUNT_VOLTAGE_LSB_UV` = 5
   
*LSB value for shunt voltage measurements in microvolts.*

### 8.59.1 Detailed Description

Header file for the [INA3221](#) triple-channel power monitor driver.

Definition in file [INA3221.h](#).

### 8.59.2 Enumeration Type Documentation

#### 8.59.2.1 `ina3221_addr_t`

`enum ina3221_addr_t`

**Enumerator**

<code>INA3221_ADDR40_GND</code>	
<code>INA3221_ADDR41_VCC</code>	
<code>INA3221_ADDR42_SDA</code>	
<code>INA3221_ADDR43_SCL</code>	

Definition at line 12 of file [INA3221.h](#).

#### 8.59.2.2 `ina3221_ch_t`

`enum ina3221_ch_t`

Enumerator

INA3221_CH1	
INA3221_CH2	
INA3221_CH3	

Definition at line 23 of file [INA3221.h](#).

### 8.59.2.3 ina3221\_reg\_t

```
enum ina3221_reg_t
```

Register addresses for [INA3221](#).

Enumerator

INA3221_REG_CONF	
INA3221_REG_CH1_SHUNTV	
INA3221_REG_CH1_BUSV	
INA3221_REG_CH2_SHUNTV	
INA3221_REG_CH2_BUSV	
INA3221_REG_CH3_SHUNTV	
INA3221_REG_CH3_BUSV	
INA3221_REG_CH1_CRIT_ALERT_LIM	
INA3221_REG_CH1_WARNING_ALERT_LIM	
INA3221_REG_CH2_CRIT_ALERT_LIM	
INA3221_REG_CH2_WARNING_ALERT_LIM	
INA3221_REG_CH3_CRIT_ALERT_LIM	
INA3221_REG_CH3_WARNING_ALERT_LIM	
INA3221_REG_SHUNTV_SUM	
INA3221_REG_SHUNTV_SUM_LIM	
INA3221_REG_MASK_ENABLE	
INA3221_REG_PWR_VALID_HI_LIM	
INA3221_REG_PWR_VALID_LO_LIM	
INA3221_REG_MANUF_ID	
INA3221_REG_DIE_ID	

Definition at line 38 of file [INA3221.h](#).

### 8.59.2.4 ina3221\_conv\_time\_t

```
enum ina3221_conv_time_t
```

Conversion time settings.

Time taken for each measurement conversion

Enumerator

INA3221_REG_CONF_CT_140US
INA3221_REG_CONF_CT_204US
INA3221_REG_CONF_CT_332US
INA3221_REG_CONF_CT_588US
INA3221_REG_CONF_CT_1100US
INA3221_REG_CONF_CT_2116US
INA3221_REG_CONF_CT_4156US
INA3221_REG_CONF_CT_8244US

Definition at line 65 of file [INA3221.h](#).

#### 8.59.2.5 ina3221\_avg\_mode\_t

```
enum ina3221_avg_mode_t
```

Averaging mode settings.

Number of samples to average for each measurement

Enumerator

INA3221_REG_CONF_AVG_1
INA3221_REG_CONF_AVG_4
INA3221_REG_CONF_AVG_16
INA3221_REG_CONF_AVG_64
INA3221_REG_CONF_AVG_128
INA3221_REG_CONF_AVG_256
INA3221_REG_CONF_AVG_512
INA3221_REG_CONF_AVG_1024

Definition at line 80 of file [INA3221.h](#).

### 8.59.3 Variable Documentation

#### 8.59.3.1 INA3221\_CH\_NUM

```
const int INA3221_CH_NUM = 3
```

Number of channels in [INA3221](#).

Definition at line 30 of file [INA3221.h](#).

#### 8.59.3.2 SHUNT\_VOLTAGE\_LSB\_UV

```
const int SHUNT_VOLTAGE_LSB_UV = 5
```

LSB value for shunt voltage measurements in microvolts.

Definition at line 32 of file [INA3221.h](#).

## 8.60 INA3221.h

[Go to the documentation of this file.](#)

```

00001 #ifndef BEASTDEVICES_INA3221_H
00002 #define BEASTDEVICES_INA3221_H
00003
00004 #include <stdint.h>
00005 #include <iostream>
00006 #include <hardware/i2c.h>
00007
00012 typedef enum {
00013     INA3221_ADDR40_GND = 0b1000000, // A0 pin -> GND
00014     INA3221_ADDR41_VCC = 0b1000001, // A0 pin -> VCC
00015     INA3221_ADDR42_SDA = 0b1000010, // A0 pin -> SDA
00016     INA3221_ADDR43_SCL = 0b1000011 // A0 pin -> SCL
00017 } ina3221_addr_t;
00018
00023 typedef enum {
00024     INA3221_CH1 = 0,
00025     INA3221_CH2,
00026     INA3221_CH3,
00027 } ina3221_ch_t;
00028
00030 const int INA3221_CH_NUM = 3;
00032 const int SHUNT_VOLTAGE_LSB_UV = 5;
00033
00034
00038 typedef enum {
00039     INA3221_REG_CONF = 0,
00040     INA3221_REG_CH1_SHUNTV,
00041     INA3221_REG_CH1_BUSV,
00042     INA3221_REG_CH2_SHUNTV,
00043     INA3221_REG_CH2_BUSV,
00044     INA3221_REG_CH3_SHUNTV,
00045     INA3221_REG_CH3_BUSV,
00046     INA3221_REG_CH1_CRIT_ALERT_LIM,
00047     INA3221_REG_CH1_WARNING_ALERT_LIM,
00048     INA3221_REG_CH2_CRIT_ALERT_LIM,
00049     INA3221_REG_CH2_WARNING_ALERT_LIM,
00050     INA3221_REG_CH3_CRIT_ALERT_LIM,
00051     INA3221_REG_CH3_WARNING_ALERT_LIM,
00052     INA3221_REG_SHUNTV_SUM,
00053     INA3221_REG_SHUNTV_SUM_LIM,
00054     INA3221_REG_MASK_ENABLE,
00055     INA3221_REG_PWR_VALID_HI_LIM,
00056     INA3221_REG_PWR_VALID_LO_LIM,
00057     INA3221_REG_MANUF_ID = 0xFE,
00058     INA3221_REG_DIE_ID = 0xFF
00059 } ina3221_reg_t;
00060
00065 typedef enum {
00066     INA3221_REG_CONF_CT_140US = 0,
00067     INA3221_REG_CONF_CT_204US,
00068     INA3221_REG_CONF_CT_332US,
00069     INA3221_REG_CONF_CT_588US,
00070     INA3221_REG_CONF_CT_1100US,
00071     INA3221_REG_CONF_CT_2116US,
00072     INA3221_REG_CONF_CT_4156US,
00073     INA3221_REG_CONF_CT_8244US
00074 } ina3221_conv_time_t;
00075
00080 typedef enum {
00081     INA3221_REG_CONF_AVG_1 = 0,
00082     INA3221_REG_CONF_AVG_4,
00083     INA3221_REG_CONF_AVG_16,
00084     INA3221_REG_CONF_AVG_64,
00085     INA3221_REG_CONF_AVG_128,
00086     INA3221_REG_CONF_AVG_256,
00087     INA3221_REG_CONF_AVG_512,
00088     INA3221_REG_CONF_AVG_1024
00089 } ina3221_avg_mode_t;
00090
00096 class INA3221 {
00097
00101     typedef struct {
00102         uint16_t mode_shunt_en:1;
00103         uint16_t mode_bus_en:1;
00104         uint16_t mode_continious_en:1;
00105         uint16_t shunt_conv_time:3;
00106         uint16_t bus_conv_time:3;
00107         uint16_t avg_mode:3;
00108         uint16_t ch3_en:1;
00109         uint16_t ch2_en:1;
00110         uint16_t ch1_en:1;
00111         uint16_t reset:1;

```

```

00112     } conf_reg_t;
00113
00117     typedef struct {
00118         uint16_t conv_ready:1;
00119         uint16_t timing_ctrl_alert:1;
00120         uint16_t pwr_valid_alert:1;
00121         uint16_t warn_alert_ch3:1;
00122         uint16_t warn_alert_ch2:1;
00123         uint16_t warn_alert_ch1:1;
00124         uint16_t shunt_sum_alert:1;
00125         uint16_t crit_alert_ch3:1;
00126         uint16_t crit_alert_ch2:1;
00127         uint16_t crit_alert_ch1:1;
00128         uint16_t crit_alert_latch_en:1;
00129         uint16_t warn_alert_latch_en:1;
00130         uint16_t shunt_sum_en_ch3:1;
00131         uint16_t shunt_sum_en_ch2:1;
00132         uint16_t shunt_sum_en_ch1:1;
00133         uint16_t reserved:1;
00134     } masken_reg_t;
00135
00136     i2c_inst_t* _i2c;
00137     // I2C address
00138     ina3221_addr_t _i2c_addr;
00139
00140     // Shunt resistance in mOhm
00141     uint32_t _shuntRes[INA3221_CH_NUM];
00142
00143     // Series filter resistance in Ohm
00144     uint32_t _filterRes[INA3221_CH_NUM];
00145
00146     // Value of Mask/Enable register.
00147     masken_reg_t _masken_reg;
00148
00149     // Reads 16 bytes from a register.
00150     void _read(ina3221_reg_t reg, uint16_t *val);
00151
00152     // Writes 16 bytes to a register.
00153     void _write(ina3221_reg_t reg, uint16_t *val);
00154
00155 public:
00156
00157     INA3221(ina3221_addr_t addr, i2c_inst_t* i2c);
00158     // Initializes INA3221
00159     bool begin();
00160
00161     // Gets a register value.
00162     uint16_t read_register(ina3221_reg_t reg);
00163
00164     // Resets INA3221
00165     void reset();
00166
00167     // Sets operating mode to power-down
00168     void set_mode_power_down();
00169
00170     // Sets operating mode to continuous
00171     void set_mode_continuous();
00172
00173     // Sets operating mode to triggered (single-shot)
00174     void set_mode_triggered();
00175
00176     // Enables shunt-voltage measurement
00177     void set_shunt_measurement_enable();
00178
00179     // Disables shunt-voltage measurement
00180     void set_shunt_measurement_disable();
00181
00182     // Enables bus-voltage measurement
00183     void set_bus_measurement_enable();
00184
00185     // Disables bus-voltage measurement
00186     void set_bus_measurement_disable();
00187
00188     // Sets averaging mode. Sets number of samples that are collected
00189     // and averaged together.
00190     void set_averaging_mode(ina3221_avg_mode_t mode);
00191
00192     // Sets bus-voltage conversion time.
00193     void set_bus_conversion_time(ina3221_conv_time_t convTime);
00194
00195     // Sets shunt-voltage conversion time.
00196     void set_shunt_conversion_time(ina3221_conv_time_t convTime);
00197
00198     // Gets manufacturer ID.
00199     // Should read 0x5449.
00200     uint16_t get_manufacturer_id();
00201

```

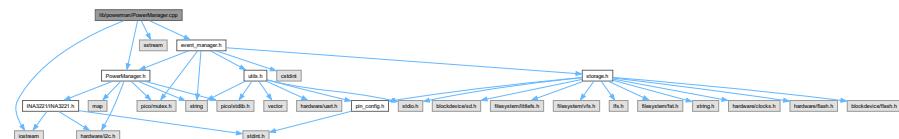
```

00202 // Gets die ID.
00203 // Should read 0x3220.
00204 uint16_t get_die_id();
00205
00206 // Gets shunt voltage in uV.
00207 int32_t get_shunt_voltage(ina3221_ch_t channel);
00208
00209 // Gets current in A.
00210 float get_current(ina3221_ch_t channel);
00211
00212 float get_current_ma(ina3221_ch_t channel);
00213
00214 // Gets bus voltage in V.
00215 float get_voltage(ina3221_ch_t channel);
00216
00217 void set_warn_alert_limit(ina3221_ch_t channel, float voltage_v);
00218 void set_crit_alert_limit(ina3221_ch_t channel, float voltage_v);
00219 void set_power_valid_limit(float voltage_upper_v, float voltage_lower_v);
00220 void enable_alerts();
00221 bool get_warn_alert(ina3221_ch_t channel);
00222 bool get_crit_alert(ina3221_ch_t channel);
00223 bool get_power_valid_alert();
00224 void set_alert_latch(bool enable);
00225 };
00226
00227 #endif

```

## 8.61 lib/powerman/PowerManager.cpp File Reference

```
#include "PowerManager.h"
#include <iostream>
#include <sstream>
#include "event_manager.h"
Include dependency graph for PowerManager.cpp:
```



## 8.62 PowerManager.cpp

[Go to the documentation of this file.](#)

```

00001 #include "PowerManager.h"
00002 #include <iostream>
00003 #include <sstream>
00004 #include "event_manager.h"
00005
00006 PowerManager::PowerManager(i2c_inst_t* i2c)
00007     : ina3221_(INA3221_ADDR40_GND, i2c) {
00008     recursive_mutex_init(&powerman_mutex_);
00009 };
00010
00011 bool PowerManager::initialize() {
00012     recursive_mutex_enter_blocking(&powerman_mutex_);
00013     initialized_ = ina3221_.begin();
00014
00015     if (initialized_) {
00016         // Set up alerts
00017         ina3221_.set_warn_alert_limit(INA3221_CH2, VOLTAGE_LOW_THRESHOLD);
00018         ina3221_.set_crit_alert_limit(INA3221_CH2, VOLTAGE_OVERCHARGE_THRESHOLD);
00019         ina3221_.set_power_valid_limit(VOLTAGE_OVERCHARGE_THRESHOLD, VOLTAGE_LOW_THRESHOLD);
00020         ina3221_.enable_alerts();
00021         ina3221_.set_alert_latch(true);
00022     }
00023

```

```

00024     recursive_mutex_exit(&powerman_mutex_);
00025     return initialized_;
00026 }
00027
00028 std::string PowerManager::read_device_ids() {
00029     if (!initialized_) return "noinit";
00030     recursive_mutex_enter_blocking(&powerman_mutex_);
00031     std::stringstream man_ss;
00032     man_ss << std::hex << ina3221_.get_manufacturer_id();
00033     std::string MAN = "MAN 0x" + man_ss.str();
00034
00035     std::stringstream die_ss;
00036     die_ss << std::hex << ina3221_.get_die_id();
00037     std::string DIE = "DIE 0x" + die_ss.str();
00038     recursive_mutex_exit(&powerman_mutex_);
00039     return MAN + " - " + DIE;
00040 }
00041
00042 float PowerManager::get_voltage_battery() {
00043     if (!initialized_) return 0.0f;
00044     recursive_mutex_enter_blocking(&powerman_mutex_);
00045     float voltage = ina3221_.get_voltage(INA3221_CH1);
00046     recursive_mutex_exit(&powerman_mutex_);
00047     return voltage;
00048 }
00049
00050 float PowerManager::get_voltage_5v() {
00051     if (!initialized_) return 0.0f;
00052     recursive_mutex_enter_blocking(&powerman_mutex_);
00053     float voltage = ina3221_.get_voltage(INA3221_CH2);
00054     recursive_mutex_exit(&powerman_mutex_);
00055     return voltage;
00056 }
00057
00058 float PowerManager::get_current_charge_usb() {
00059     if (!initialized_) return 0.0f;
00060     recursive_mutex_enter_blocking(&powerman_mutex_);
00061     float current = ina3221_.get_current_ma(INA3221_CH1);
00062     recursive_mutex_exit(&powerman_mutex_);
00063     return current;
00064 }
00065
00066 float PowerManager::get_current_draw() {
00067     if (!initialized_) return 0.0f;
00068     recursive_mutex_enter_blocking(&powerman_mutex_);
00069     float current = ina3221_.get_current_ma(INA3221_CH2);
00070     recursive_mutex_exit(&powerman_mutex_);
00071     return current;
00072 }
00073
00074 float PowerManager::get_current_charge_solar() {
00075     if (!initialized_) return 0.0f;
00076     recursive_mutex_enter_blocking(&powerman_mutex_);
00077     float current = ina3221_.get_current_ma(INA3221_CH3);
00078     recursive_mutex_exit(&powerman_mutex_);
00079     return current;
00080 }
00081
00082 float PowerManager::get_current_charge_total() {
00083     if (!initialized_) return 0.0f;
00084     recursive_mutex_enter_blocking(&powerman_mutex_);
00085     float current = ina3221_.get_current_ma(INA3221_CH1) + ina3221_.get_current_ma(INA3221_CH3);
00086     recursive_mutex_exit(&powerman_mutex_);
00087     return current;
00088 }
00089
00090 void PowerManager::configure(const std::map<std::string, std::string>& config) {
00091     if (!initialized_) return;
00092     recursive_mutex_enter_blocking(&powerman_mutex_);
00093
00094     if (config.find("operating_mode") != config.end()) {
00095         if (config.at("operating_mode") == "continuous") {
00096             ina3221_.set_mode_continuous();
00097         }
00098     }
00099
00100    if (config.find("averaging_mode") != config.end()) {
00101        int avg_mode = std::stoi(config.at("averaging_mode"));
00102        switch(avg_mode) {
00103            case 1:
00104                ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_1);
00105                break;
00106            case 4:
00107                ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_4);
00108                break;
00109            case 16:
00110                ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_16);
00111        }
00112    }
}

```

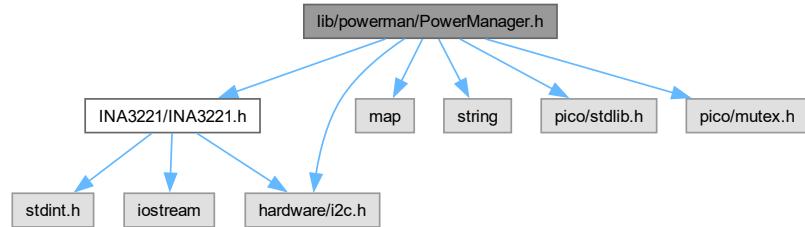
```

00111         break;
00112     default:
00113         ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_16);
00114     }
00115 }
00116 recursive_mutex_exit(&powerman_mutex_);
00117 }
00118
00119 bool PowerManager::is_charging_solar() {
00120     if (!initialized_) return false;
00121     recursive_mutex_enter_blocking(&powerman_mutex_);
00122     bool active = get_current_charge_solar() > SOLAR_CURRENT_THRESHOLD;
00123     recursive_mutex_exit(&powerman_mutex_);
00124     return active;
00125 }
00126
00127 bool PowerManager::is_charging_usb() {
00128     if (!initialized_) return false;
00129     recursive_mutex_enter_blocking(&powerman_mutex_);
00130     bool connected = get_current_charge_usb() > USB_CURRENT_THRESHOLD;
00131     recursive_mutex_exit(&powerman_mutex_);
00132     return connected;
00133 }
00134
00135 bool PowerManager::check_power_alerts() {
00136     if (!initialized_) return false;
00137
00138     recursive_mutex_enter_blocking(&powerman_mutex_);
00139
00140     bool status_changed = false;
00141
00142     // Check warning alert (low battery)
00143     if (ina3221_.get_warn_alert(INA3221_CH2)) {
00144         EventEmitter::emit(EventGroup::POWER, PowerEvent::LOW_BATTERY);
00145         status_changed = true;
00146     }
00147
00148     // Check critical alert (overcharge)
00149     if (ina3221_.get_crit_alert(INA3221_CH2)) {
00150         EventEmitter::emit(EventGroup::POWER, PowerEvent::OVERCHARGE);
00151         status_changed = true;
00152     }
00153
00154     // Check power valid alert
00155     if (ina3221_.get_power_valid_alert()) {
00156         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_NORMAL);
00157         status_changed = true;
00158     }
00159
00160     recursive_mutex_exit(&powerman_mutex_);
00161     return status_changed;
00162 }
```

## 8.63 lib/powerman/PowerManager.h File Reference

```
#include "INA3221/INA3221.h"
#include <map>
#include <string>
#include <hardware/i2c.h>
#include "pico/stdlib.h"
#include "pico/mutex.h"
```

Include dependency graph for PowerManager.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [PowerManager](#)

## 8.64 PowerManager.h

[Go to the documentation of this file.](#)

```

00001 #ifndef POWER_MANAGER_H
00002 #define POWER_MANAGER_H
00003
00004 #include "INA3221/INA3221.h"
00005 #include <map>
00006 #include <string>
00007 #include <hardware/i2c.h>
00008 #include "pico/stdlib.h"
00009 #include "pico/mutex.h"
00010
00011 class PowerManager {
00012 public:
00013     PowerManager(i2c_inst_t* i2c);
00014     bool initialize();
00015     std::string read_device_ids();
00016     float get_current_charge_solar();
00017     float get_current_charge_usb();
00018     float get_current_charge_total();
00019     float get_current_draw();
00020     float get_voltage_battery();
00021     float get_voltage_5v();
00022     void configure(const std::map<std::string, std::string>& config);
00023     bool is_charging_solar();
00024     bool is_charging_usb();
00025     bool check_power_alerts();
00026
00027     static constexpr float SOLAR_CURRENT_THRESHOLD = 50.0f; // mA
00028     static constexpr float USB_CURRENT_THRESHOLD = 50.0f; // mA
00029     static constexpr float VOLTAGE_LOW_THRESHOLD = 4.6f; // V
00030     static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f; // V
00031     static constexpr float FALL_RATE_THRESHOLD = -0.02f; // V/sample
00032     static constexpr int FALLING_TREND_REQUIRED = 3; // samples
00033
00034
  
```

```

00035 private:
00036     INA3221 ina3221_;
00037     bool initialized_;
00038     recursive_mutex_t powerman_mutex_;
00039     bool charging_solar_active_ = false;
00040     bool charging_usb_active_ = false;
00041 };
00042
00043 #endif // POWER_MANAGER_H

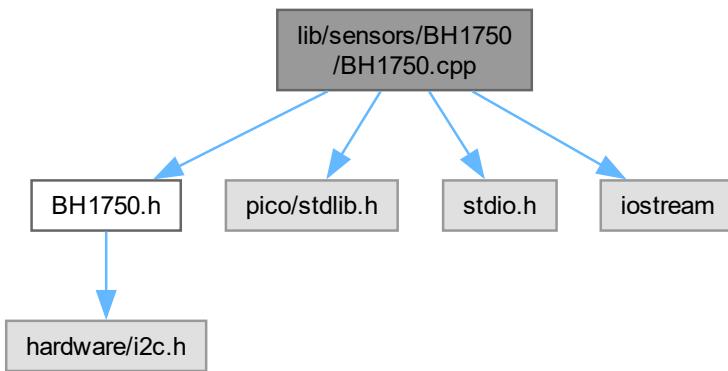
```

## 8.65 lib/sensors/BH1750/BH1750.cpp File Reference

```

#include "BH1750.h"
#include "pico/stl.h"
#include <stdio.h>
#include <iostream>
Include dependency graph for BH1750.cpp:

```



## 8.66 BH1750.cpp

[Go to the documentation of this file.](#)

```

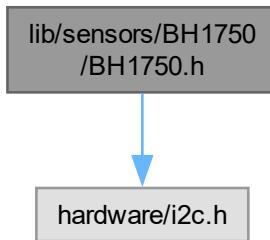
00001 #include "BH1750.h"
00002 #include "pico/stl.h"
00003 #include <stdio.h>
00004 #include <iostream>
00005
00006 BH1750::BH1750(uint8_t addr) : _i2c_addr(addr) {}
00007
00008 bool BH1750::begin(Mode mode) {
00009     write8(static_cast<uint8_t>(Mode::POWER_ON));
00010     write8(static_cast<uint8_t>(Mode::RESET));
00011     configure(mode);
00012     configure(BH1750::Mode::POWER_ON);
00013     uint8_t check = 0;
00014     uint8_t cmd = 0x10; // Continuously H-Resolution Mode
00015     if (_i2c_write_blocking(i2c1, _i2c_addr, &cmd, 1, false) == 1) {
00016         std::cout << "BH1750 sensor found at 0x" << std::hex << (int)_i2c_addr << std::endl;
00017         return true;
00018     }
00019     return false;
00020 }
00021
00022 void BH1750::configure(Mode mode) {

```

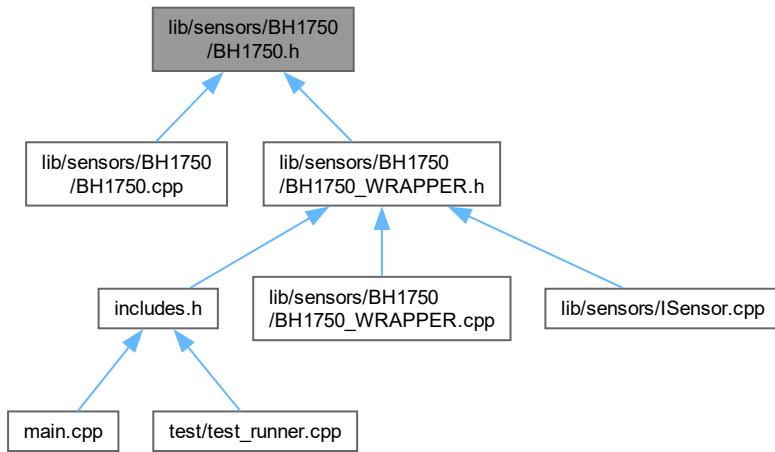
```
00023     uint8_t modeVal = static_cast<uint8_t>(mode);
00024     switch (mode) {
00025         case Mode::CONTINUOUS_HIGH_RES_MODE:
00026         case Mode::CONTINUOUS_HIGH_RES_MODE_2:
00027         case Mode::CONTINUOUS_LOW_RES_MODE:
00028         case Mode::ONE_TIME_HIGH_RES_MODE:
00029         case Mode::ONE_TIME_HIGH_RES_MODE_2:
00030         case Mode::ONE_TIME_LOW_RES_MODE:
00031             write8(modeVal);
00032             sleep_ms(10);
00033             break;
00034     default:
00035         printf("Invalid measurement mode\n");
00036         break;
00037     }
00038 }
00039
00040 float BH1750::get_light_level() {
00041     uint8_t buffer[2];
00042     i2c_read_blocking(i2c_default, _i2c_addr, buffer, 2, false);
00043     uint16_t level = (buffer[0] << 8) | buffer[1];
00044
00045     float lux = static_cast<float>(level) / 1.2f;
00046     return lux;
00047 }
00048
00049 void BH1750::write8(uint8_t data) {
00050     uint8_t buf[1] = {data};
00051     i2c_write_blocking(i2c_default, _i2c_addr, buf, 1, false);
00052 }
```

## 8.67 lib/sensors/BH1750/BH1750.h File Reference

```
#include "hardware/i2c.h"
Include dependency graph for BH1750.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [BH1750](#)

## Macros

- `#define _BH1750_DEVICE_ID 0xE1`
- `#define _BH1750_MTREG_MIN 31`
- `#define _BH1750_MTREG_MAX 254`
- `#define _BH1750_DEFAULT_MTREG 69`

### 8.67.1 Macro Definition Documentation

#### 8.67.1.1 `_BH1750_DEVICE_ID`

```
#define _BH1750_DEVICE_ID 0xE1
```

Definition at line [7](#) of file [BH1750.h](#).

#### 8.67.1.2 `_BH1750_MTREG_MIN`

```
#define _BH1750_MTREG_MIN 31
```

Definition at line [8](#) of file [BH1750.h](#).

### 8.67.1.3 \_BH1750\_MTREG\_MAX

```
#define _BH1750_MTREG_MAX 254
```

Definition at line 9 of file [BH1750.h](#).

### 8.67.1.4 \_BH1750\_DEFAULT\_MTREG

```
#define _BH1750_DEFAULT_MTREG 69
```

Definition at line 10 of file [BH1750.h](#).

## 8.68 BH1750.h

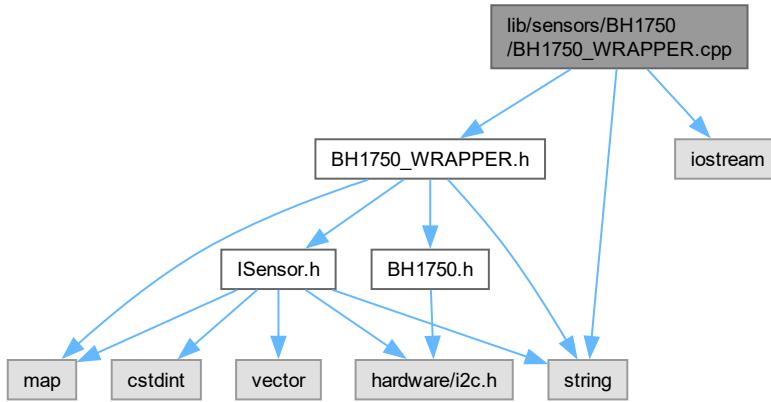
[Go to the documentation of this file.](#)

```
00001 #ifndef __BH1750_H__
00002 #define __BH1750_H__
00003
00004 #include "hardware/i2c.h"
00005
00006 // Define constants
00007 #define _BH1750_DEVICE_ID 0xE1 // Correct content of WHO_AM_I register
00008 #define _BH1750_MTREG_MIN 31
00009 #define _BH1750_MTREG_MAX 254
00010 #define _BH1750_DEFAULT_MTREG 69
00011
00012 class BH1750 {
00013 public:
00014     // Scoped enum for measurement modes
00015     enum class Mode : uint8_t {
00016         UNCONFIGURED_POWER_DOWN = 0x00,
00017         POWER_ON = 0x01,
00018         RESET = 0x07,
00019         CONTINUOUS_HIGH_RES_MODE = 0x10,
00020         CONTINUOUS_HIGH_RES_MODE_2 = 0x11,
00021         CONTINUOUS_LOW_RES_MODE = 0x13,
00022         ONE_TIME_HIGH_RES_MODE = 0x20,
00023         ONE_TIME_HIGH_RES_MODE_2 = 0x21,
00024         ONE_TIME_LOW_RES_MODE = 0x23
00025     };
00026
00027     BH1750(uint8_t addr = 0x23);
00028     bool begin(Mode mode = Mode::CONTINUOUS_HIGH_RES_MODE);
00029     void configure(Mode mode);
00030     float get_light_level();
00031
00032 private:
00033     void write8(uint8_t data);
00034     uint8_t _i2c_addr;
00035 };
00036
00037 #endif // __BH1750_H__
```

## 8.69 lib/sensors/BH1750/BH1750\_WRAPPER.cpp File Reference

```
#include "BH1750_WRAPPER.h"
#include <string>
```

```
#include <iostream>
Include dependency graph for BH1750_WRAPPER.cpp:
```



## 8.70 BH1750\_WRAPPER.cpp

[Go to the documentation of this file.](#)

```

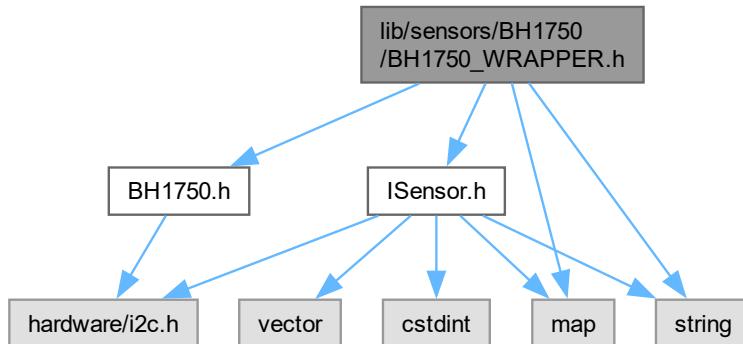
00001 // BH1750Wrapper.cpp
00002 #include "BH1750_WRAPPER.h"
00003 #include <string>
00004 #include <iostream>
00005
00006 BH1750Wrapper::BH1750Wrapper() {
00007     sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE);
00008 }
00009
00010 bool BH1750Wrapper::init() {
00011     initialized_ = sensor_.begin();
00012     return initialized_;
00013 }
00014
00015 float BH1750Wrapper::read_data(SensorDataTypeIdentifier type) {
00016     if (type == SensorDataTypeIdentifier::LIGHT_LEVEL) {
00017         return sensor_.get_light_level();
00018     }
00019     return 0.0f;
00020 }
00021
00022 bool BH1750Wrapper::is_initialized() const {
00023     return initialized_;
00024 }
00025
00026 SensorType BH1750Wrapper::get_type() const {
00027     return SensorType::LIGHT;
00028 }
00029
00030 bool BH1750Wrapper::configure(const std::map<std::string, std::string>& config) {
00031     for (const auto& [key, value] : config) {
00032         if (key == "measurement_mode") {
00033             if (value == "continuously_high_resolution") {
00034                 sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE);
00035             }
00036             else if (value == "continuously_high_resolution_2") {
00037                 sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE_2);
00038             }
00039             else if (value == "continuously_low_resolution") {
00040                 sensor_.configure(BH1750::Mode::CONTINUOUS_LOW_RES_MODE);
00041             }
00042             else if (value == "one_time_high_resolution") {
00043                 sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE);
00044             }
00045             else if (value == "one_time_high_resolution_2") {
  
```

```

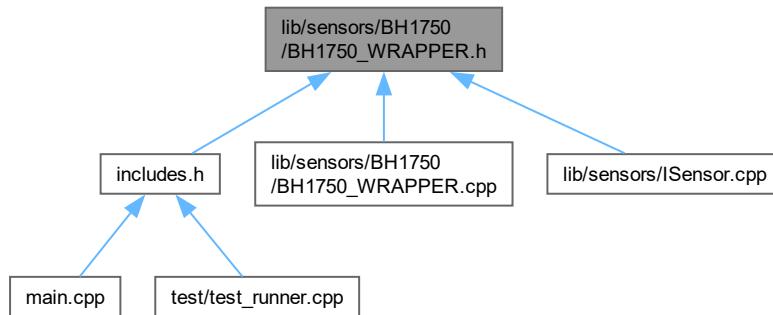
00046         sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE_2);
00047     }
00048     else if (value == "one_time_low_resolution") {
00049         sensor_.configure(BH1750::Mode::ONE_TIME_LOW_RES_MODE);
00050     }
00051     else {
00052         std::cerr << "[BH1750Wrapper] Unknown measurement_mode value: " << value << std::endl;
00053         return false;
00054     }
00055     // Handle additional configuration keys here
00056     else {
00057         std::cerr << "[BH1750Wrapper] Unknown configuration key: " << key << std::endl;
00058         return false;
00059     }
00060 }
00061
00062     return true;
00063 }
```

## 8.71 lib/sensors/BH1750/BH1750\_WRAPPER.h File Reference

```
#include "ISensor.h"
#include "BH1750.h"
#include <map>
#include <string>
Include dependency graph for BH1750_WRAPPER.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [BH1750Wrapper](#)

## 8.72 BH1750\_WRAPPER.h

[Go to the documentation of this file.](#)

```

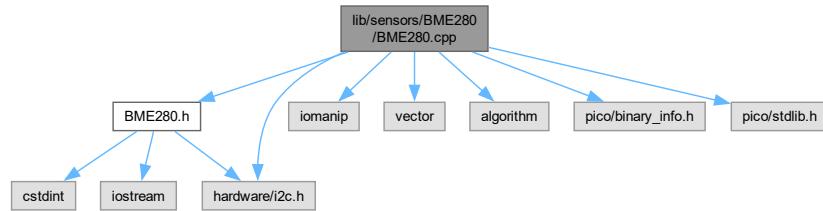
00001 #ifndef BH1750_WRAPPER_H
00002 #define BH1750_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "BH1750.h"
00006 #include <map>
00007 #include <string>
00008
00009 class BH1750Wrapper : public ISensor {
00010 private:
00011     BH1750 sensor_;
00012     bool initialized_ = false;
00013
00014 public:
00015     BH1750Wrapper();
00016     int get_i2c_addr();
00017     bool init() override;
00018     float read_data(SensorDataTypeIdentifier type) override;
00019     bool is_initialized() const override;
00020     SensorType get_type() const override;
00021
00022     bool configure(const std::map<std::string, std::string>& config);
00023
00024     uint8_t get_address() const override {
00025         return 0x23;
00026     }
00027 };
00028
00029 #endif // BH1750_WRAPPER_H
  
```

## 8.73 lib/sensors/BME280/BME280.cpp File Reference

```

#include "BME280.h"
#include <iomanip>
#include <vector>
  
```

```
#include <algorithm>
#include "hardware/i2c.h"
#include "pico/binary_info.h"
#include "pico/stdc.h"
Include dependency graph for BME280.cpp:
```



## 8.74 BME280.cpp

[Go to the documentation of this file.](#)

```
00001 // BME280.cpp
00002
00003 #include "BME280.h"
00004
00005 #include <iomanip>
00006 #include <vector>
00007 #include <algorithm>
00008 #include "hardware/i2c.h"
00009 #include "pico/binary_info.h"
00010 #include "pico/stdc.h"
00011
00012 // BME280 (BME280) Class Implementation
00013
00014 BME280::BME280(i2c_inst_t* i2cPort, uint8_t address)
00015     : i2c_port(i2cPort), device_addr(address), calib_params{}, initialized_(false), t_fine(0) {
00016 }
00017
00018 bool BME280::init() {
00019     if (!i2c_port) {
00020         std::cerr << "Invalid I2C port.\n";
00021         return false;
00022     }
00023
00024     // Check device ID to confirm it's a BME280
00025     uint8_t reg = 0xD0; // Chip ID register
00026     uint8_t chip_id = 0;
00027     int ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
00028     if (ret != 1) {
00029         std::cerr << "Failed to write to BME280.\n";
00030         return false;
00031     }
00032     ret = i2c_read_blocking(i2c_port, device_addr, &chip_id, 1, false);
00033     if (ret != 1) {
00034         std::cerr << "Failed to read chip ID from BME280.\n";
00035         return false;
00036     }
00037     if (chip_id != 0x60) {
00038         std::cerr << "Device is not a BME280.\n";
00039         return false;
00040     }
00041
00042     // Configure sensor
00043     if (!configure_sensor()) {
00044         std::cerr << "Failed to configure BME280 sensor.\n";
00045         return false;
00046     }
00047
00048     // Retrieve calibration parameters
00049     if (!get_calibration_parameters()) {
00050         std::cerr << "Failed to retrieve calibration parameters.\n";
00051         return false;
00052     }
```

```

00053     initialized_ = true;
00054     std::cout << "BME280 sensor initialized_ successfully.\n";
00055     return true;
00056 }
00058
00059 void BME280::reset() {
00060     uint8_t buf[2] = { REG_RESET, 0xB6 };
00061     int ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00062     if (ret != 2) {
00063         std::cerr << "Failed to reset BME280 sensor.\n";
00064     }
00065     sleep_ms(10); // Wait for reset to complete
00066 }
00067
00068 bool BME280::read_raw_all(int32_t* temperature, int32_t* pressure, int32_t* humidity) {
00069     if (!initialized_) {
00070         std::cerr << "BME280 not initialized_.\n";
00071         return false;
00072     }
00073
00074     // Define the starting register address
00075     uint8_t start_reg = REG_PRESSURE_MSB;
00076     // Total bytes to read: 3 (pressure) + 3 (temperature) + 2 (humidity) = 8
00077     uint8_t buf[8] = {0};
00078
00079     // Write the starting register address
00080     int ret = i2c_write_blocking(i2c_port, device_addr, &start_reg, 1, true);
00081     if (ret != 1) {
00082         std::cerr << "Failed to write starting register address to BME280.\n";
00083         return false;
00084     }
00085
00086     // Read data
00087     ret = i2c_read_blocking(i2c_port, device_addr, buf, 8, false);
00088     if (ret != 8) {
00089         std::cerr << "Failed to read data from BME280.\n";
00090         return false;
00091     }
00092
00093     // Combine bytes to form raw values
00094     *pressure = ((int32_t)buf[0] << 12) | ((int32_t)buf[1] << 4) | ((int32_t)(buf[2] >> 4));
00095     *temperature = ((int32_t)buf[3] << 12) | ((int32_t)buf[4] << 4) | ((int32_t)(buf[5] >> 4));
00096     *humidity = ((int32_t)buf[6] << 8) | (int32_t)buf[7];
00097
00098     return true;
00099 }
00100
00101 float BME280::convert_temperature(int32_t temp_raw) const {
00102     int32_t var1, var2;
00103     var1 = (((temp_raw >> 3) - ((int32_t)calib_params.dig_t1 << 1))) * ((int32_t)calib_params.dig_t2)
00104     » 11;
00105     var2 = (((((temp_raw >> 4) - ((int32_t)calib_params.dig_t1)) * ((temp_raw >> 4) -
00106     ((int32_t)calib_params.dig_t1)) » 12) * ((int32_t)calib_params.dig_t3)) » 14;
00107     t_fine = var1 + var2;
00108     float T = (t_fine * 5 + 128) » 8;
00109     return T / 100.0f;
00110 }
00111
00112 float BME280::convert_pressure(int32_t pressure_raw) const {
00113     int64_t var1, var2, p;
00114     var1 = ((int64_t)t_fine) - 128000;
00115     var2 = var1 * var1 * (int64_t)calib_params.dig_p6;
00116     var2 = var2 + ((var1 * (int64_t)calib_params.dig_p5) » 17);
00117     var2 = var2 + (((int64_t)calib_params.dig_p4) » 35);
00118     var1 = ((var1 * var1 * (int64_t)calib_params.dig_p3) » 8) + ((var1 * (int64_t)calib_params.dig_p2)
00119     » 12);
00120     var1 = (((int64_t)1 << 47) + var1) * ((int64_t)calib_params.dig_p1) » 33;
00121
00122     if (var1 == 0) {
00123         return 0.0f; // avoid exception caused by division by zero
00124     }
00125     p = 1048576 - pressure_raw;
00126     p = ((p << 31) - var2) * 3125) / var1;
00127     var1 = (((int64_t)calib_params.dig_p9) * (p » 13) * (p » 13)) » 25;
00128     var2 = (((int64_t)calib_params.dig_p8) * p) » 19;
00129
00130     p = ((p + var1 + var2) » 8) + (((int64_t)calib_params.dig_p7) » 4);
00131     return (float)p / 25600.0f; // in hPa
00132 }
00133
00134 float BME280::convert_humidity(int32_t humidity_raw) const {
00135     int32_t v_x1_u32r;
00136     v_x1_u32r = t_fine - 76800;
00137     v_x1_u32r = (((humidity_raw << 14) - ((int32_t)calib_params.dig_h4 << 20) -
00138     ((int32_t)calib_params.dig_h5 * v_x1_u32r)) + 16384) » 15) *
00139     (((((v_x1_u32r * (int32_t)calib_params.dig_h6) » 10) * (((v_x1_u32r *

```

```

00136     (int32_t)calib_params.dig_h3) >> 11) + 32768)) >> 10) + 2097152) *
00137     (int32_t)calib_params.dig_h2 + 8192) >> 14));
00138 v_x1_u32r = std::max(v_x1_u32r, (int32_t)0);
00139 v_x1_u32r = std::min(v_x1_u32r, (int32_t)419430400);
00140 float h = v_x1_u32r >> 12;
00141 } 00142
00143 bool BME280::get_calibration_parameters() {
00144     // Read temperature and pressure calibration data (0x88 to 0xA1)
00145     uint8_t calib_data[26];
00146     uint8_t reg = REG_DIG_T1_LSB;
00147     int ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
00148     if (ret != 1) {
00149         std::cerr << "Failed to write to BME280.\n";
00150         return false;
00151     }
00152     ret = i2c_read_blocking(i2c_port, device_addr, calib_data, 26, false);
00153     if (ret != 26) {
00154         std::cerr << "Failed to read calibration data from BME280.\n";
00155         return false;
00156     }
00157
00158     // Parse temperature calibration data
00159     calib_params.dig_t1 = (uint16_t)(calib_data[1] << 8 | calib_data[0]);
00160     calib_params.dig_t2 = (int16_t)(calib_data[3] << 8 | calib_data[2]);
00161     calib_params.dig_t3 = (int16_t)(calib_data[5] << 8 | calib_data[4]);
00162
00163     // Parse pressure calibration data
00164     calib_params.dig_p1 = (uint16_t)(calib_data[7] << 8 | calib_data[6]);
00165     calib_params.dig_p2 = (int16_t)(calib_data[9] << 8 | calib_data[8]);
00166     calib_params.dig_p3 = (int16_t)(calib_data[11] << 8 | calib_data[10]);
00167     calib_params.dig_p4 = (int16_t)(calib_data[13] << 8 | calib_data[12]);
00168     calib_params.dig_p5 = (int16_t)(calib_data[15] << 8 | calib_data[14]);
00169     calib_params.dig_p6 = (int16_t)(calib_data[17] << 8 | calib_data[16]);
00170     calib_params.dig_p7 = (int16_t)(calib_data[19] << 8 | calib_data[18]);
00171     calib_params.dig_p8 = (int16_t)(calib_data[21] << 8 | calib_data[20]);
00172     calib_params.dig_p9 = (int16_t)(calib_data[23] << 8 | calib_data[22]);
00173
00174     calib_params.dig_h1 = calib_data[25];
00175
00176     // Read humidity calibration data (0xE1 to 0xE7)
00177     reg = 0xE1;
00178     ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
00179     if (ret != 1) {
00180         std::cerr << "Failed to write to BME280 for humidity calibration.\n";
00181         return false;
00182     }
00183
00184     uint8_t hum_calib_data[7];
00185     ret = i2c_read_blocking(i2c_port, device_addr, hum_calib_data, 7, false);
00186     if (ret != 7) {
00187         std::cerr << "Failed to read humidity calibration data from BME280.\n";
00188         return false;
00189     }
00190
00191     // Parse humidity calibration data
00192     calib_params.dig_h2 = (int16_t)(hum_calib_data[1] << 8 | hum_calib_data[0]);
00193     calib_params.dig_h3 = hum_calib_data[2];
00194     calib_params.dig_h4 = (int16_t)((hum_calib_data[3] << 4) | (hum_calib_data[4] & 0x0F));
00195     calib_params.dig_h5 = (int16_t)((hum_calib_data[5] << 4) | (hum_calib_data[4] >> 4));
00196     calib_params.dig_h6 = (int8_t)hum_calib_data[6];
00197
00198     return true;
00199 }
00200
00201 bool BME280::configure_sensor() {
00202     uint8_t buf[2];
00203
00204     // Set humidity oversampling (must be set before ctrl_meas)
00205     buf[0] = REG_CTRL_HUM;
00206     buf[1] = 0x05; // Humidity oversampling x16
00207     int ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00208     if (ret != 2) {
00209         std::cerr << "Failed to write CTRL_HUM to BME280.\n";
00210         return false;
00211     }
00212
00213     // Write config register
00214     buf[0] = REG_CONFIG;
00215     buf[1] = 0x00; // Default settings
00216     ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00217     if (ret != 2) {
00218         std::cerr << "Failed to write CONFIG to BME280.\n";
00219         return false;
00220     }
00221

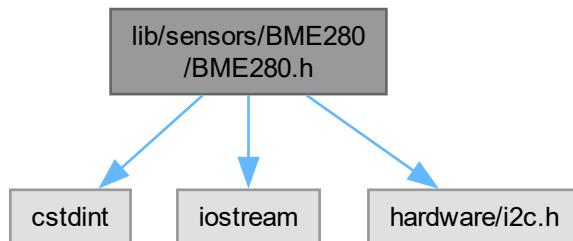
```

```

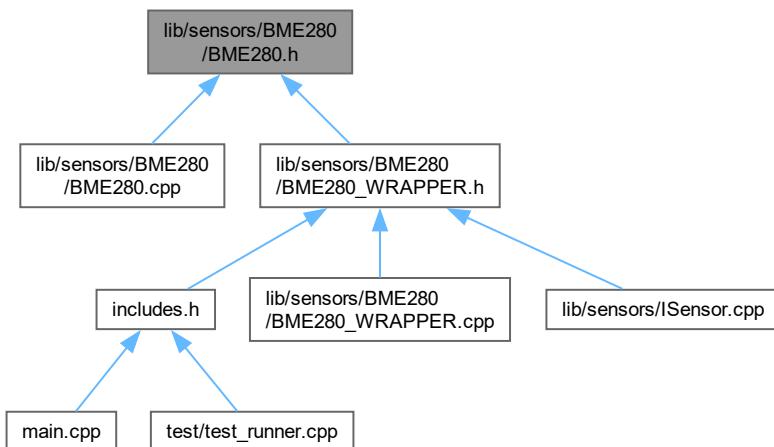
00222 // Write ctrl_meas register
00223 buf[0] = REG_CTRL_MEAS;
00224 buf[1] = 0xB7; // Temp and pressure oversampling x16, normal mode
00225 ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00226 if (ret !=2) {
00227     std::cerr << "Failed to write CTRL_MEAS to BME280.\n";
00228     return false;
00229 }
00230
00231 return true;
00232 }
```

## 8.75 lib/sensors/BME280/BME280.h File Reference

```
#include <cstdint>
#include <iostream>
#include "hardware/i2c.h"
Include dependency graph for BME280.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct **BME280CalibParam**
- class **BME280**

## 8.76 BME280.h

[Go to the documentation of this file.](#)

```

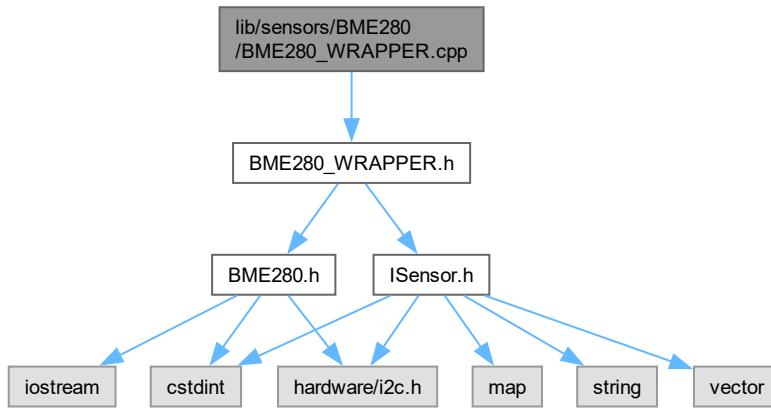
00001 // BME280.h
00002
00003 #ifndef BME280_H
00004 #define BME280_H
00005
00006 #include <cstdint>
00007 #include <iostream>
00008 #include "hardware/i2c.h"
00009
00010 // Calibration parameters structure
00011 struct BME280CalibParam {
00012     // Temperature parameters
00013     uint16_t dig_t1;
00014     int16_t dig_t2;
00015     int16_t dig_t3;
00016
00017     // Pressure parameters
00018     uint16_t dig_p1;
00019     int16_t dig_p2;
00020     int16_t dig_p3;
00021     int16_t dig_p4;
00022     int16_t dig_p5;
00023     int16_t dig_p6;
00024     int16_t dig_p7;
00025     int16_t dig_p8;
00026     int16_t dig_p9;
00027
00028     // Humidity parameters
00029     uint8_t dig_h1;
00030     int16_t dig_h2;
00031     uint8_t dig_h3;
00032     int16_t dig_h4;
00033     int16_t dig_h5;
00034     int8_t dig_h6;
00035 };
00036
00037 // BME280 Class Definition
00038 class BME280 {
00039 public:
00040     // I2C Address Options
00041     static constexpr uint8_t ADDR_SDO_LOW = 0x76;
00042     static constexpr uint8_t ADDR_SDO_HIGH = 0x77;
00043
00044     // Constructor
00045     BME280(i2c_inst_t* i2cPort, uint8_t address = ADDR_SDO_LOW);
00046
00047     // Initialize the sensor
00048     bool init();
00049
00050     // Reset the sensor
00051     void reset();
00052
00053     // Read all raw data: temperature, pressure, and humidity
00054     bool read_raw_all(int32_t* temperature, int32_t* pressure, int32_t* humidity);
00055
00056     // Convert raw data to actual values
00057     float convert_temperature(int32_t temp_raw) const;
00058     float convert_pressure(int32_t pressure_raw) const;
00059     float convert_humidity(int32_t humidity_raw) const;
00060
00061 private:
00062     // Configure the sensor
00063     bool configure_sensor();
00064
00065     // Retrieve calibration parameters from the sensor
00066     bool get_calibration_parameters();
00067
00068     // I2C port and device address
00069     i2c_inst_t* i2c_port;
00070     uint8_t device_addr;
00071

```

```
00072 // Calibration parameters
00073 BME280CalibParam calib_params;
00074
00075 // Initialization status
00076 bool initialized_;
00077
00078 // Fine temperature parameter needed for compensation
00079 mutable int32_t t_fine;
00080
00081 // Register Definitions
00082 static constexpr uint8_t REG_CONFIG = 0xF5;
00083 static constexpr uint8_t REG_CTRL_MEAS = 0xF4;
00084 static constexpr uint8_t REG_CTRL_HUM = 0xF2;
00085 static constexpr uint8_t REG_RESET = 0xE0;
00086
00087 static constexpr uint8_t REG_PRESSURE_MSB = 0xF7;
00088 static constexpr uint8_t REG_TEMPERATURE_MSB = 0xFA;
00089 static constexpr uint8_t REG_HUMIDITY_MSB = 0xFD;
00090
00091 // Calibration Registers
00092 static constexpr uint8_t REG_DIG_T1_LSB = 0x88;
00093 static constexpr uint8_t REG_DIG_T1_MSB = 0x89;
00094 static constexpr uint8_t REG_DIG_T2_LSB = 0x8A;
00095 static constexpr uint8_t REG_DIG_T2_MSB = 0x8B;
00096 static constexpr uint8_t REG_DIG_T3_LSB = 0x8C;
00097 static constexpr uint8_t REG_DIG_T3_MSB = 0x8D;
00098
00099 static constexpr uint8_t REG_DIG_P1_LSB = 0x8E;
00100 static constexpr uint8_t REG_DIG_P1_MSB = 0x8F;
00101 static constexpr uint8_t REG_DIG_P2_LSB = 0x90;
00102 static constexpr uint8_t REG_DIG_P2_MSB = 0x91;
00103 static constexpr uint8_t REG_DIG_P3_LSB = 0x92;
00104 static constexpr uint8_t REG_DIG_P3_MSB = 0x93;
00105 static constexpr uint8_t REG_DIG_P4_LSB = 0x94;
00106 static constexpr uint8_t REG_DIG_P4_MSB = 0x95;
00107 static constexpr uint8_t REG_DIG_P5_LSB = 0x96;
00108 static constexpr uint8_t REG_DIG_P5_MSB = 0x97;
00109 static constexpr uint8_t REG_DIG_P6_LSB = 0x98;
00110 static constexpr uint8_t REG_DIG_P6_MSB = 0x99;
00111 static constexpr uint8_t REG_DIG_P7_LSB = 0x9A;
00112 static constexpr uint8_t REG_DIG_P7_MSB = 0x9B;
00113 static constexpr uint8_t REG_DIG_P8_LSB = 0x9C;
00114 static constexpr uint8_t REG_DIG_P8_MSB = 0x9D;
00115 static constexpr uint8_t REG_DIG_P9_LSB = 0x9E;
00116 static constexpr uint8_t REG_DIG_P9_MSB = 0x9F;
00117
00118 // Humidity Calibration Registers
00119 static constexpr uint8_t REG_DIG_H1 = 0xA1;
00120 static constexpr uint8_t REG_DIG_H2 = 0xE1;
00121 static constexpr uint8_t REG_DIG_H3 = 0xE3;
00122 static constexpr uint8_t REG_DIG_H4 = 0xE4;
00123 static constexpr uint8_t REG_DIG_H5 = 0xE5;
00124 static constexpr uint8_t REG_DIG_H6 = 0xE7;
00125
00126 // Number of calibration parameters to read
00127 static constexpr size_t NUM_CALIB_PARAMS = 24;
00128 };
00129
00130 #endif // BME280_H
```

## 8.77 lib/sensors/BME280/BME280\_WRAPPER.cpp File Reference

```
#include "BME280_WRAPPER.h"
Include dependency graph for BME280_WRAPPER.cpp:
```



## 8.78 BME280\_WRAPPER.cpp

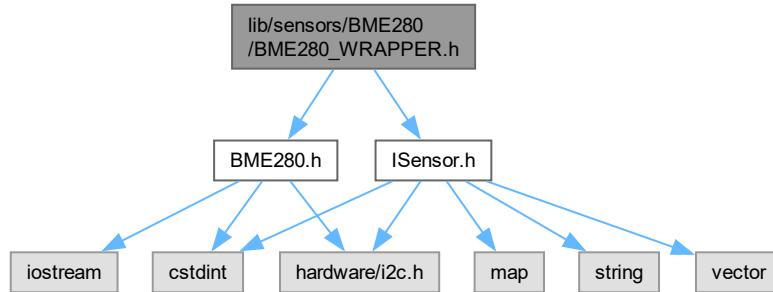
[Go to the documentation of this file.](#)

```

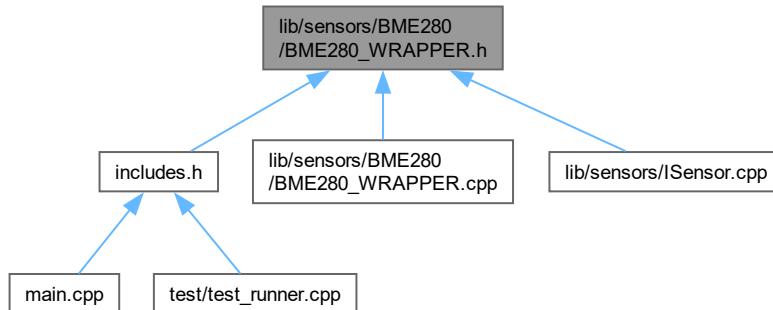
00001 #include "BME280_WRAPPER.h"
00002
00003 BME280Wrapper::BME280Wrapper(i2c_inst_t* i2c) : sensor_(i2c) {}
00004
00005 bool BME280Wrapper::init() {
00006     initialized_ = sensor_.init();
00007     return initialized_;
00008 }
00009
00010 float BME280Wrapper::read_data(SensorDataTypeIdentifier type) {
00011     int32_t temp_raw, pressure_raw, humidity_raw;
00012     sensor_.read_raw_all(&temp_raw, &pressure_raw, &humidity_raw);
00013
00014     switch(type) {
00015         case SensorDataTypeIdentifier::TEMPERATURE:
00016             return sensor_.convert_temperature(temp_raw);
00017         case SensorDataTypeIdentifier::PRESSURE:
00018             return sensor_.convert_pressure(pressure_raw);
00019         case SensorDataTypeIdentifier::HUMIDITY:
00020             return sensor_.convert_humidity(humidity_raw);
00021         default:
00022             return 0.0f;
00023     }
00024 }
00025
00026 bool BME280Wrapper::is_initialized() const {
00027     return initialized_;
00028 }
00029
00030 SensorType BME280Wrapper::get_type() const {
00031     return SensorType::ENVIRONMENT;
00032 }
00033
00034 bool BME280Wrapper::configure(const std::map<std::string, std::string>& config) {
00035     return true;
00036 }
```

## 8.79 lib/sensors/BME280/BME280\_WRAPPER.h File Reference

```
#include "ISensor.h"
#include "BME280.h"
Include dependency graph for BME280_WRAPPER.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [BME280Wrapper](#)

## 8.80 BME280\_WRAPPER.h

[Go to the documentation of this file.](#)

```
00001 // BME280_WRAPPER.h
00002 #ifndef BME280_WRAPPER_H
00003 #define BME280_WRAPPER_H
00004
00005 #include "ISensor.h"
00006 #include "BME280.h"
00007
```

```

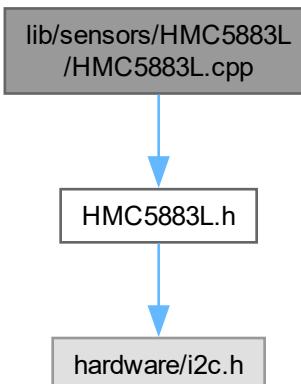
00008 class BME280Wrapper : public ISensor {
00009     private:
0010         BME280 sensor_;
0011         bool initialized_ = false;
0012
0013     public:
0014         BME280Wrapper(i2c_inst_t* i2c);
0015
0016         bool init() override;
0017         float read_data(SensorDataTypeIdentifier type) override;
0018         bool is_initialized() const override;
0019         SensorType get_type() const override;
0020         bool configure(const std::map<std::string, std::string>& config) override;
0021
0022         uint8_t get_address() const override {
0023             return 0x76;
0024         }
0025
0026     };
0027
0028 #endif // BME280_WRAPPER_H

```

## 8.81 lib/sensors/HMC5883L/HMC5883L.cpp File Reference

#include "HMC5883L.h"

Include dependency graph for HMC5883L.cpp:



## 8.82 HMC5883L.cpp

[Go to the documentation of this file.](#)

```

00001 #include "HMC5883L.h"
00002
00003 HMC5883L::HMC5883L(i2c_inst_t* i2c, uint8_t address) : i2c(i2c), address(address) {}
00004
00005 bool HMC5883L::init() {
00006     // Continuous measurement mode, 15Hz data output rate
00007     if (!write_register(0x00, 0x70)) return false;
00008     if (!write_register(0x01, 0x20)) return false;
00009     if (!write_register(0x02, 0x00)) return false;
00010     return true;
00011 }
00012
00013 bool HMC5883L::read(int16_t& x, int16_t& y, int16_t& z) {

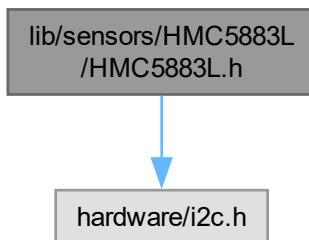
```

```
00014     uint8_t buffer[6];
00015     if (!read_register(0x03, buffer, 6)) return false;
00016
00017     x = (buffer[0] < 8) | buffer[1];
00018     z = (buffer[2] < 8) | buffer[3];
00019     y = (buffer[4] < 8) | buffer[5];
00020
00021     if (x > 32767) x -= 65536;
00022     if (y > 32767) y -= 65536;
00023     if (z > 32767) z -= 65536;
00024
00025     return true;
00026 }
00027
00028 bool HMC5883L::write_register(uint8_t reg, uint8_t value) {
00029     uint8_t buffer[2] = {reg, value};
00030     return i2c_write_blocking(i2c, address, buffer, 2, false) == 2;
00031 }
00032
00033 bool HMC5883L::read_register(uint8_t reg, uint8_t* buffer, size_t length) {
00034     if (i2c_write_blocking(i2c, address, &reg, 1, true) != 1) return false;
00035     return i2c_read_blocking(i2c, address, buffer, length, false) == length;
00036 }
```

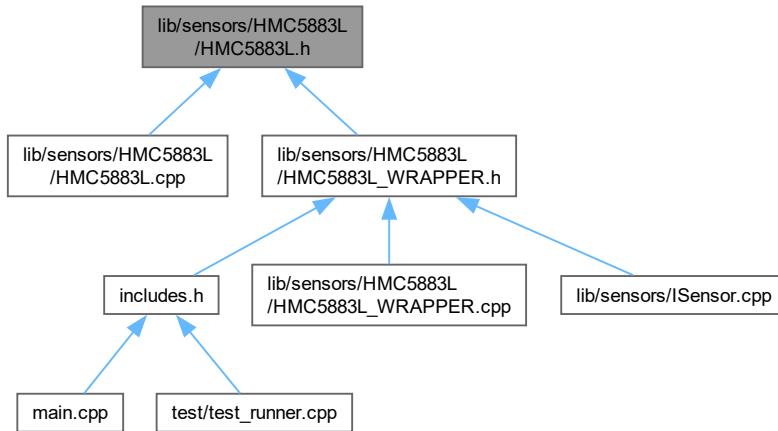
## 8.83 lib/sensors/HMC5883L/HMC5883L.h File Reference

#include "hardware/i2c.h"

Include dependency graph for HMC5883L.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [HMC5883L](#)

## 8.84 HMC5883L.h

[Go to the documentation of this file.](#)

```

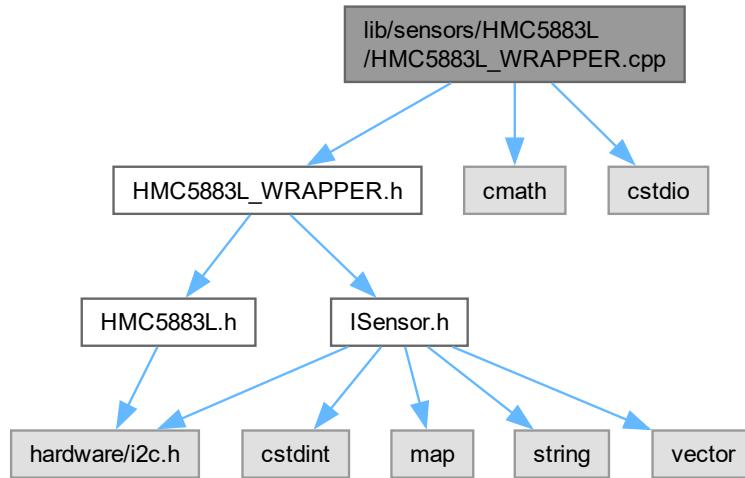
00001 #ifndef HMC5883L_H
00002 #define HMC5883L_H
00003
00004 #include "hardware/i2c.h"
00005
00006 class HMC5883L {
00007 public:
00008     HMC5883L(i2c_inst_t* i2c, uint8_t address = 0x0D);
00009     bool init();
00010     bool read(int16_t& x, int16_t& y, int16_t& z);
00011
00012 private:
00013     i2c_inst_t* i2c;
00014     uint8_t address;
00015
00016     bool write_register(uint8_t reg, uint8_t value);
00017     bool read_register(uint8_t reg, uint8_t* buffer, size_t length);
00018 };
00019
00020 #endif
  
```

## 8.85 lib/sensors/HMC5883L/HMC5883L\_WRAPPER.cpp File Reference

```

#include "HMC5883L_WRAPPER.h"
#include <cmath>
  
```

```
#include <cstdio>
Include dependency graph for HMC5883L_WRAPPER.cpp:
```



## 8.86 HMC5883L\_WRAPPER.cpp

[Go to the documentation of this file.](#)

```

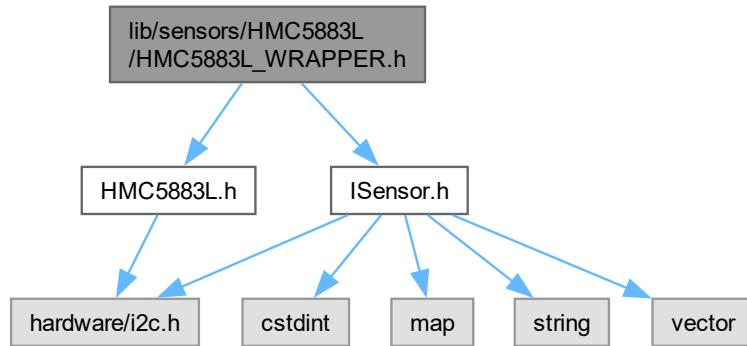
00001 #include "HMC5883L_WRAPPER.h"
00002 #include <cmath>
00003 #include <cstdio>
00004
00005 HMC5883LWrapper::HMC5883LWrapper(i2c_inst_t* i2c) : sensor_(i2c), initialized_(false) {}
00006
00007 bool HMC5883LWrapper::init() {
00008     initialized_ = sensor_.init();
00009     return initialized_;
00010 }
00011
00012 float HMC5883LWrapper::read_data(SensorDataTypeIdentifier type) {
00013     if (!initialized_) return 0.0f;
00014
00015     int16_t x, y, z;
00016     if (!sensor_.read(x, y, z)) return 0.0f;
00017
00018     const float LSB_TO_UT = 100.0 / 1090.0;
00019     float x_uT = x * LSB_TO_UT;
00020     float y_uT = y * LSB_TO_UT;
00021     float z_uT = z * LSB_TO_UT;
00022
00023     switch (type) {
00024         case SensorDataTypeIdentifier::MAG_FIELD_X:
00025             return x_uT;
00026         case SensorDataTypeIdentifier::MAG_FIELD_Y:
00027             return y_uT;
00028         case SensorDataTypeIdentifier::MAG_FIELD_Z:
00029             return z_uT;
00030         default:
00031             return 0.0f;
00032     }
00033 }
00034
00035 bool HMC5883LWrapper::is_initialized() const {
00036     return initialized_;
00037 }
00038
00039 SensorType HMC5883LWrapper::get_type() const {
00040     return SensorType::MAGNETOMETER;
  
```

```

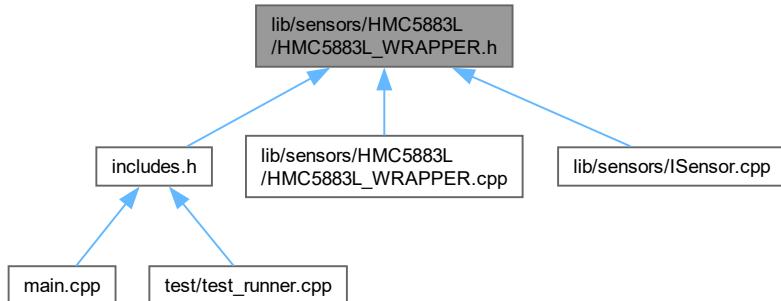
00041 }
00042
00043 bool HMC5883LWrapper::configure(const std::map<std::string, std::string>& config) {
00044     // Configuration logic if needed
00045     return true;
00046 }
```

## 8.87 lib/sensors/HMC5883L/HMC5883L\_WRAPPER.h File Reference

```
#include "ISensor.h"
#include "HMC5883L.h"
Include dependency graph for HMC5883L_WRAPPER.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [HMC5883LWrapper](#)

## 8.88 HMC5883L\_WRAPPER.h

[Go to the documentation of this file.](#)

```

00001 #ifndef HMC5883L_WRAPPER_H
00002 #define HMC5883L_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "HMC5883L.h"
00006
00007 class HMC5883LWrapper : public ISensor {
00008 public:
00009     HMC5883LWrapper(i2c_inst_t* i2c);
00010     bool init() override;
00011     float read_data(SensorDataTypeIdentifier type) override;
00012     bool is_initialized() const override;
00013     SensorType get_type() const override;
00014     bool configure(const std::map<std::string, std::string>& config) override;
00015
00016     uint8_t get_address() const override {
00017         return 0x0D;
00018     }
00019
00020 private:
00021     HMC5883L sensor_;
00022     bool initialized_;
00023 };
00024
00025 #endif

```

## 8.89 lib/sensors/ISensor.cpp File Reference

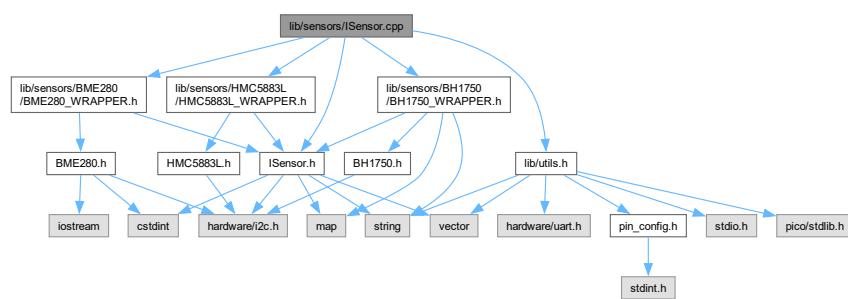
Implements the [SensorWrapper](#) class for managing different sensor types.

```

#include "ISensor.h"
#include "lib/sensors/BH1750/BH1750_WRAPPER.h"
#include "lib/sensors/BME280/BME280_WRAPPER.h"
#include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
#include "lib/utils.h"

Include dependency graph for ISensor.cpp:

```



### 8.89.1 Detailed Description

Implements the [SensorWrapper](#) class for managing different sensor types.

This file provides the implementation for initializing, configuring, and reading data from various sensors.

Definition in file [ISensor.cpp](#).

## 8.90 ISensor.cpp

[Go to the documentation of this file.](#)

```

00001 // ISensor.cpp
00002 #include "ISensor.h"
00003 #include "lib/sensors/BH1750/BH1750_WRAPPER.h"
00004 #include "lib/sensors/BME280/BME280_WRAPPER.h"
00005 #include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
00006 #include "lib/utils.h"
00007
00014
00019
00024 SensorWrapper& SensorWrapper::get_instance() {
00025     static SensorWrapper instance;
00026     return instance;
00027 }
00028
00029
00033 SensorWrapper::SensorWrapper() = default;
00034
00035
00042 bool SensorWrapper::sensor_init(SensorType type, i2c_inst_t* i2c) {
00043     switch(type) {
00044         case SensorType::LIGHT:
00045             sensors[type] = new BH1750Wrapper();
00046             break;
00047         case SensorType::ENVIRONMENT:
00048             sensors[type] = new BME280Wrapper(i2c);
00049             break;
00050         case SensorType::IMU:
00051             //sensors[type] = new MPU6050Wrapper(i2c);
00052             break;
00053         case SensorType::MAGNETOMETER:
00054             sensors[type] = new HMC5883LWrapper(i2c);
00055             break;
00056     }
00057     return sensors[type]->init();
00058 }
00059
00060
00067 bool SensorWrapper::sensor_configure(SensorType type, const std::map<std::string, std::string>& config) {
00068     auto it = sensors.find(type);
00069     if (it != sensors.end() && it->second->is_initialized()) {
00070         return it->second->configure(config);
00071     }
00072     std::cerr << "Sensor not initialized or not found: " << static_cast<int>(type) << std::endl;
00073     return false;
00074 }
00075
00076
00083 float SensorWrapper::sensor_read_data(SensorType sensorType, SensorDataTypeIdentifier dataType) {
00084     auto it = sensors.find(sensorType);
00085     if (it != sensors.end() && it->second->is_initialized()) {
00086         return it->second->read_data(dataType);
00087     }
00088     return 0.0f;
00089 }
00090
00091
00096 std::vector<std::pair<SensorType, uint8_t>› SensorWrapper::get_available_sensors() {
00097     std::vector<std::pair<SensorType, uint8_t>› available_sensors;
00098
00099     for (const auto& sensor_pair : sensors) {
00100         if (sensor_pair.second->is_initialized()) {
00101             available_sensors.push_back({sensor_pair.first, sensor_pair.second->get_address()});
00102         }
00103     }
00104
00105     return available_sensors;
00106 }
00107
00108
00114 std::vector<std::pair<SensorType, uint8_t>› SensorWrapper::scan_connected_sensors(i2c_inst_t* i2c) {
00115     std::vector<std::pair<SensorType, uint8_t>› connected_sensors;
00116
00117     // Define the address ranges to check for each sensor type
00118     struct SensorAddressInfo {
00119         SensorType type;
00120         std::vector<uint8_t> addresses;
00121     };
00122
00123     std::vector<SensorAddressInfo> sensor_addresses = {
00124         {SensorType::LIGHT, {0x23, 0x5C}},           // BH1750 addresses
00125         {SensorType::ENVIRONMENT, {0x76, 0x77}},      // BME280 addresses

```

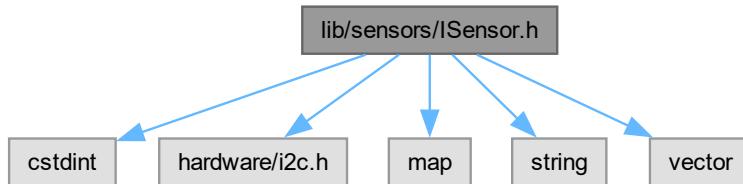
```

00126     {SensorType::MAGNETOMETER, {0x0D, 0x1E}},           // HMC5883L addresses
00127     {SensorType::IMU, {0x68, 0x69}}                      // MPU6050 addresses
00128 };
00129
00130 // Buffer for receiving ACK/NACK
00131 uint8_t rxdata;
00132
00133 for (const auto& sensor_info : sensor_addresses) {
00134     for (uint8_t addr : sensor_info.addresses) {
00135         // Try to read a byte from the device to see if it ACKs
00136         int result = i2c_read_blocking(i2c, addr, &rxdata, 1, false);
00137         if (result >= 0) {
00138             // We received an ACK, so the device exists
00139             connected_sensors.push_back({sensor_info.type, addr});
00140             uart_print("Found sensor at address 0x" + std::to_string(addr),
00141             VerbosityLevel::DEBUG);
00142         }
00143     }
00144 }
00145 return connected_sensors;
00146 }
00147

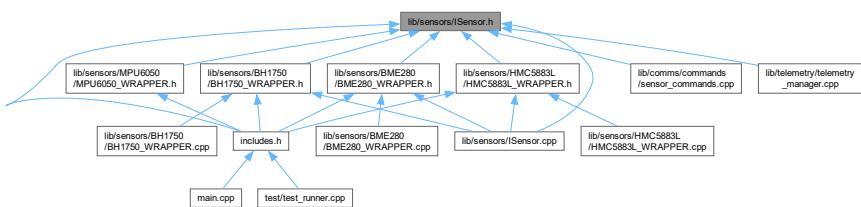
```

## 8.91 lib/sensors/ISensor.h File Reference

```
#include <cstdint>
#include "hardware/i2c.h"
#include <map>
#include <string>
#include <vector>
Include dependency graph for ISensor.h:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class `ISensor`
- class `SensorWrapper`

*Manages different sensor types and provides a unified interface for accessing sensor data.*

## Enumerations

- enum class `SensorType` { `LIGHT` , `ENVIRONMENT` , `MAGNETOMETER` , `IMU` }
- enum class `SensorDataTypelIdentifier` {  
`LIGHT_LEVEL` , `TEMPERATURE` , `PRESSURE` , `HUMIDITY` ,  
`MAG_FIELD_X` , `MAG_FIELD_Y` , `MAG_FIELD_Z` , `GYRO_X` ,  
`GYRO_Y` , `GYRO_Z` , `ACCEL_X` , `ACCEL_Y` ,  
`ACCEL_Z` }

### 8.91.1 Enumeration Type Documentation

#### 8.91.1.1 SensorType

```
enum class SensorType [strong]
```

##### Enumerator

<code>LIGHT</code>	
<code>ENVIRONMENT</code>	
<code>MAGNETOMETER</code>	
<code>IMU</code>	

Definition at line 11 of file `ISensor.h`.

#### 8.91.1.2 SensorDataTypelIdentifier

```
enum class SensorDataTypelIdentifier [strong]
```

##### Enumerator

<code>LIGHT_LEVEL</code>	
<code>TEMPERATURE</code>	
<code>PRESSURE</code>	
<code>HUMIDITY</code>	
<code>MAG_FIELD_X</code>	
<code>MAG_FIELD_Y</code>	
<code>MAG_FIELD_Z</code>	
<code>GYRO_X</code>	
<code>GYRO_Y</code>	
<code>GYRO_Z</code>	
<code>ACCEL_X</code>	
<code>ACCEL_Y</code>	
<code>ACCEL_Z</code>	

Definition at line 18 of file `ISensor.h`.

## 8.92 ISensor.h

[Go to the documentation of this file.](#)

```

00001 // ISensor.h
00002 #ifndef ISENSOR_H
00003 #define ISENSOR_H
00004
00005 #include <cstdint>
00006 #include "hardware/i2c.h"
00007 #include <map>
00008 #include <string>
00009 #include <vector>
00010
00011 enum class SensorType {
00012     LIGHT,           // BH1750
00013     ENVIRONMENT,    // BME280
00014     MAGNETOMETER,   // HMC5883L
00015     IMU,             // MPU6050
00016 };
00017
00018 enum class SensorDataTypeIdentifier {
00019     LIGHT_LEVEL,
00020     TEMPERATURE,
00021     PRESSURE,
00022     HUMIDITY,
00023     MAG_FIELD_X,
00024     MAG_FIELD_Y,
00025     MAG_FIELD_Z,
00026     GYRO_X,
00027     GYRO_Y,
00028     GYRO_Z,
00029     ACCEL_X,
00030     ACCEL_Y,
00031     ACCEL_Z
00032 };
00033
00034 class ISensor {
00035 public:
00036     virtual ~ISensor() = default;
00037     virtual bool init() = 0;
00038     virtual float read_data(SensorDataTypeIdentifier type) = 0;
00039     virtual bool is_initialized() const = 0;
00040     virtual SensorType get_type() const = 0;
00041     virtual bool configure(const std::map<std::string, std::string>& config) = 0;
00042     virtual uint8_t get_address() const = 0;
00043 };
00044
00045 class SensorWrapper {
00046 public:
00047     static SensorWrapper& get_instance();
00048     bool sensor_init(SensorType type, i2c_inst_t* i2c = nullptr);
00049     bool sensor_configure(SensorType type, const std::map<std::string, std::string>& config);
00050     float sensor_read_data(SensorType sensorType, SensorDataTypeIdentifier dataType);
00051     ISensor* get_sensor(SensorType type);
00052
00053     std::vector<std::pair<SensorType, uint8_t>> scan_connected_sensors(i2c_inst_t* i2c);
00054     std::vector<std::pair<SensorType, uint8_t>> get_available_sensors();
00055
00056 private:
00057     std::map<SensorType, ISensor*> sensors;
00058     SensorWrapper();
00059 };
00060
00061 #endif // ISENSOR_H

```

## 8.93 lib/sensors/MPU6050/MPU6050.cpp File Reference

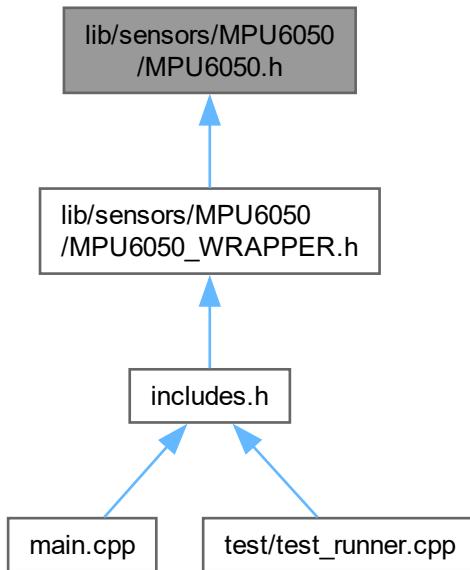
### 8.94 MPU6050.cpp

[Go to the documentation of this file.](#)

```
00001
```

## 8.95 lib/sensors/MPU6050/MPU6050.h File Reference

This graph shows which files directly or indirectly include this file:



## 8.96 MPU6050.h

[Go to the documentation of this file.](#)

00001

## 8.97 lib/sensors/MPU6050/MPU6050\_WRAPPER.cpp File Reference

## 8.98 MPU6050\_WRAPPER.cpp

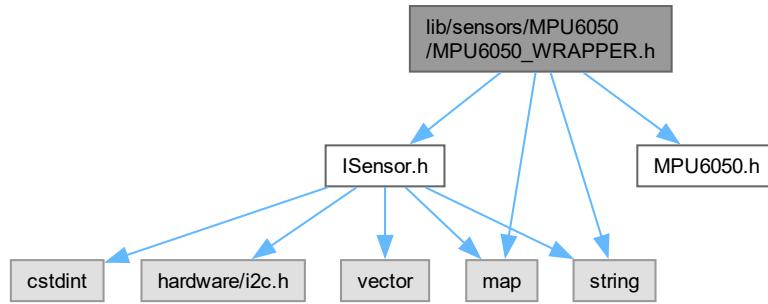
[Go to the documentation of this file.](#)

00001

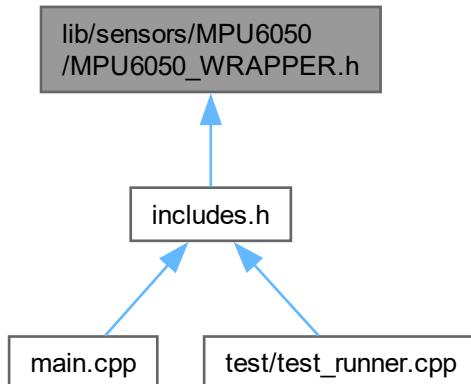
## 8.99 lib/sensors/MPU6050/MPU6050\_WRAPPER.h File Reference

```
#include "ISensor.h"
#include "MPU6050.h"
#include <map>
```

```
#include <string>
Include dependency graph for MPU6050_WRAPPER.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- class [MPU6050Wrapper](#)

## 8.100 MPU6050\_WRAPPER.h

[Go to the documentation of this file.](#)

```
00001 #ifndef BH1750_WRAPPER_H
00002 #define BH1750_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "MPU6050.h"
00006 #include <map>
```

```

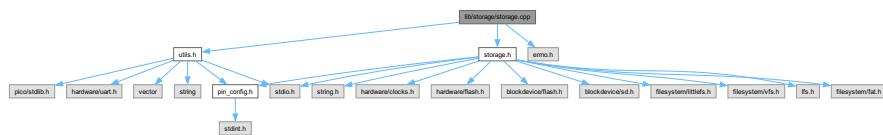
00007 #include <string>
00008
00009 class MPU6050Wrapper : public ISensor {
00010 private:
00011     MPU6050 sensor_;
00012     bool initialized_ = false;
00013
00014 public:
00015     MPU6050Wrapper();
00016
00017     bool init() override;
00018     float read_data(SensorDataTypeIdentifier type) override;
00019     bool is_initialized() const override;
00020     SensorType get_type() const override;
00021
00022     bool configure(const std::map<std::string, std::string>& config);
00023 };
00024
00025 #endif

```

## 8.101 lib/storage/storage.cpp File Reference

Implements file system operations for the Kubisat firmware.

```
#include "storage.h"
#include "errno.h"
#include "utils.h"
Include dependency graph for storage.cpp:
```



### Functions

- `bool fs_init (void)`  
*Initializes the file system on the SD card.*

### Variables

- `bool sd_card_mounted = false`

#### 8.101.1 Detailed Description

Implements file system operations for the Kubisat firmware.

This file contains functions for initializing the file system, opening, writing, reading, and closing files.

Definition in file [storage.cpp](#).

## 8.101.2 Function Documentation

### 8.101.2.1 fs\_init()

```
bool fs_init (
    void )
```

Initializes the file system on the SD card.

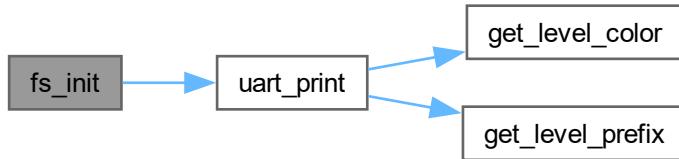
#### Returns

True if initialization was successful, false otherwise.

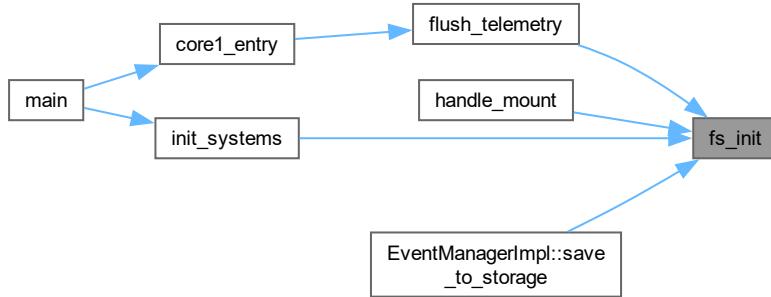
Mounts the littlefs file system on the SD card. If mounting fails, it formats the SD card with littlefs and then attempts to mount again.

Definition at line 25 of file [storage.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.101.3 Variable Documentation

#### 8.101.3.1 sd\_card\_mounted

```
bool sd_card_mounted = false
```

Definition at line 17 of file [storage.cpp](#).

## 8.102 storage.cpp

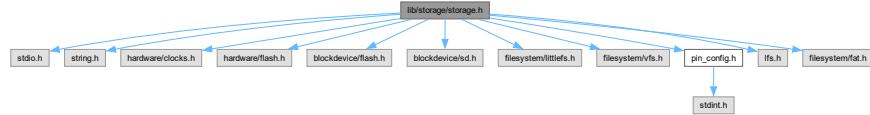
[Go to the documentation of this file.](#)

```
00001 /*
00002  * Copyright 2024, Hiroyuki OYAMA. All rights reserved.
00003  *
00004  * SPDX-License-Identifier: BSD-3-Clause
00005 */
00006 #include "storage.h"
00007 #include "errno.h"
00008 #include "utils.h"
00009
00010
00011 bool sd_card_mounted = false;
00012
00013
00014 bool fs_init(void) {
00015     sd_card_mounted = false;
00016     uart_print("fs_init littlefs on SD card", VerbosityLevel::DEBUG);
00017     blockdevice_t *sd = blockdevice_sd_create(SD_SPI_PORT,
00018                                              SD_MOSI_PIN,
00019                                              SD_MISO_PIN,
00020                                              SD_SCK_PIN,
00021                                              SD_CS_PIN,
00022                                              24 * MHZ,
00023                                              false);
00024     filesystem_t *fat = filesystem_fat_create();
00025
00026     std::string status_string;
00027     int err = fs_mount("/", fat, sd);
00028     if (err == -1) {
00029         status_string = "Formatting / with FAT";
00030         uart_print(status_string, VerbosityLevel::WARNING);
00031         err = fs_format(fat, sd);
00032         if (err == -1) {
00033             status_string = "fs_format error: " + std::string(strerror(errno));
00034             uart_print(status_string, VerbosityLevel::ERROR);
00035             return false;
00036         }
00037         err = fs_mount("/", fat, sd);
00038         if (err == -1) {
00039             status_string = "fs_mount error: " + std::string(strerror(errno));
00040             uart_print(status_string, VerbosityLevel::ERROR);
00041             return false;
00042         }
00043     }
00044     sd_card_mounted = true;
00045     return true;
00046 }
```

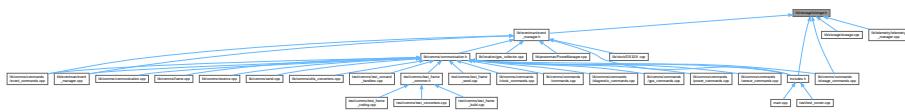
## 8.103 lib/storage/storage.h File Reference

```
#include <stdio.h>
#include <string.h>
#include <hardware/clocks.h>
#include <hardware/flash.h>
#include "blockdevice/flash.h"
#include "blockdevice/sd.h"
#include "filesystem/littlefs.h"
```

```
#include "filesystem/vfs.h"
#include "pin_config.h"
#include "lfs.h"
#include "filesystem/fat.h"
Include dependency graph for storage.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [FileHandle](#)

## Functions

- bool [fs\\_init](#) (void)
 

*Initializes the file system on the SD card.*
- [FileHandle fs\\_open\\_file](#) (const char \*filename, const char \*mode)
- ssize\_t [fs\\_write\\_file](#) (FileHandle &handle, const void \*buffer, size\_t size)
- ssize\_t [fs\\_read\\_file](#) (FileHandle &handle, void \*buffer, size\_t size)
- bool [fs\\_close\\_file](#) (FileHandle &handle)
- bool [fs\\_file\\_exists](#) (const char \*filename)

## Variables

- bool [sd\\_card\\_mounted](#)

## 8.103.1 Function Documentation

### 8.103.1.1 [fs\\_init\(\)](#)

```
bool fs_init (
    void )
```

Initializes the file system on the SD card.

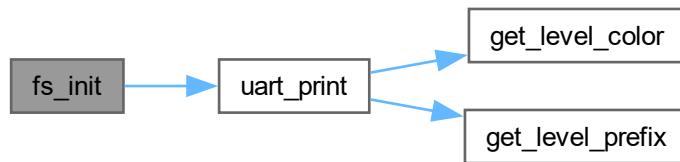
**Returns**

True if initialization was successful, false otherwise.

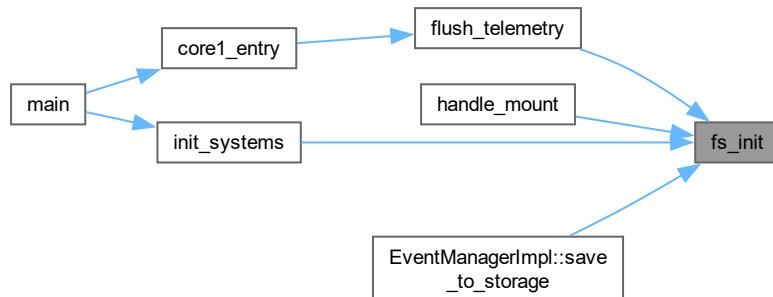
Mounts the littlefs file system on the SD card. If mounting fails, it formats the SD card with littlefs and then attempts to mount again.

Definition at line 25 of file [storage.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

**8.103.1.2 fs\_open\_file()**

```
FileHandle fs_open_file (
    const char * filename,
    const char * mode)
```

**8.103.1.3 fs\_write\_file()**

```
ssize_t fs_write_file (
    FileHandle & handle,
    const void * buffer,
    size_t size)
```

**8.103.1.4 fs\_read\_file()**

```
ssize_t fs_read_file (
    FileHandle & handle,
    void * buffer,
    size_t size)
```

**8.103.1.5 fs\_close\_file()**

```
bool fs_close_file (
    FileHandle & handle)
```

**8.103.1.6 fs\_file\_exists()**

```
bool fs_file_exists (
    const char * filename)
```

**8.103.2 Variable Documentation****8.103.2.1 sd\_card\_mounted**

```
bool sd_card_mounted [extern]
```

Definition at line 17 of file [storage.cpp](#).

**8.104 storage.h**

[Go to the documentation of this file.](#)

```
00001 #ifndef STORAGE_H
00002 #define STORAGE_H
00003
00004 #include <stdio.h>
00005 #include <string.h>
00006 #include <hardware/clocks.h>
00007 #include <hardware/flash.h>
00008 #include "blockdevice/flash.h"
00009 #include "blockdevice/sd.h"
00010 #include "filesystem/littlefs.h"
00011 #include "filesystem/vfs.h"
00012 #include "pin_config.h"
00013 #include "lfs.h"
00014 #include "filesystem/fat.h"
00015
00016
00017 extern bool sd_card_mounted;
00018
00019 struct FileHandle {
00020     int fd;
00021     bool is_open;
00022 };
00023
00024 bool fs_init(void);
00025 FileHandle fs_open_file(const char* filename, const char* mode);
00026 ssize_t fs_write_file(FileHandle& handle, const void* buffer, size_t size);
00027 ssize_t fs_read_file(FileHandle& handle, void* buffer, size_t size);
00028 bool fs_close_file(FileHandle& handle);
00029 bool fs_file_exists(const char* filename);
00030
00031 #endif
```

```
00032
00033 // void example_file_operations() {
00034 //     // Open a file for writing
00035 //     FileHandle log_file = fs_open_file("/log.txt", "w");
00036 //     if (!log_file.is_open) {
00037 //         uartPrint("Failed to open log file");
00038 //         return;
00039 //     }
00040
00041 //     // Write some data
00042 //     const char* message = "Hello, World!\n";
00043 //     ssize_t written = fs_write_file(log_file, message, strlen(message));
00044 //     if (written < 0) {
00045 //         uartPrint("Failed to write to log file");
00046 //     }
00047
00048 //     // Close the file
00049 //     fs_close_file(log_file);
00050
00051 //     // Open file for reading
00052 //     log_file = fs_open_file("/log.txt", "r");
00053 //     if (!log_file.is_open) {
00054 //         uartPrint("Failed to open log file for reading");
00055 //         return;
00056 //     }
00057
00058 //     // Read the data
00059 //     char buffer[128];
00060 //     ssize_t bytes_read = fs_read_file(log_file, buffer, sizeof(buffer) - 1);
00061 //     if (bytes_read > 0) {
00062 //         buffer[bytes_read] = '\0'; // Null terminate the string
00063 //         uartPrint(buffer);
00064 //     }
00065
00066 //     // Close the file
00067 //     fs_close_file(log_file);
00068 // }
```

## 8.105 lib/telemetry/telemetry\_manager.cpp File Reference

Implementation of telemetry collection and storage functionality.

```
#include "telemetry_manager.h"
#include "utils.h"
#include "storage.h"
#include "PowerManager.h"
#include "ISensor.h"
#include "DS3231.h"
#include <deque>
#include <mutex>
#include <iomanip>
#include <sstream>
#include <cstdio>
```

Include dependency graph for telemetry\_manager.cpp:



## Classes

- struct TelemetryRecord

*Structure representing a single telemetry data point.*

## Macros

- `#define TELEMETRY_CSV_PATH "/telemetry.csv"`  
*Path to the telemetry CSV file on storage media.*
- `#define DEFAULT_SAMPLE_INTERVAL_MS 2000`  
*Default interval between telemetry samples in milliseconds (2 seconds)*
- `#define DEFAULT_FLUSH_THRESHOLD 10`  
*Default number of records to collect before flushing to storage.*
- `#define TELEMETRY_BUFFER_SIZE 20`  
*Circular buffer for telemetry records.*

## Functions

- `bool telemetry_init ()`  
*Initialize the telemetry system.*
- `bool collect_telemetry ()`  
*Collect telemetry data from sensors and power subsystems.*
- `bool flush_telemetry ()`  
*Save buffered telemetry data to storage.*
- `bool is_telemetry_collection_time (uint32_t current_time, uint32_t &last_collection_time)`  
*Check if it's time to collect telemetry based on interval.*
- `bool is_telemetry_flush_time (uint32_t &collection_counter)`  
*Check if it's time to flush telemetry buffer based on count.*
- `uint32_t get_telemetry_sample_interval ()`  
*Get the current sample interval in milliseconds.*
- `void set_telemetry_sample_interval (uint32_t interval_ms)`  
*Set the telemetry sample interval.*
- `uint32_t get_telemetry_flush_threshold ()`  
*Get the number of records before flushing to storage.*
- `void set_telemetry_flush_threshold (uint32_t records)`  
*Set the number of records before flushing to storage.*

## Variables

- `PowerManager powerManager`
- `DS3231 systemClock`
- `NMEAData nmea_data`
- `static uint32_t sample_interval_ms = DEFAULT_SAMPLE_INTERVAL_MS`  
*Current sampling interval in milliseconds.*
- `static uint32_t flush_threshold = DEFAULT_FLUSH_THRESHOLD`  
*Current flush threshold (number of records that triggers a flush)*
- `static TelemetryRecord telemetry_buffer [TELEMETRY_BUFFER_SIZE]`
- `static size_t telemetry_buffer_count = 0`
- `static size_t telemetry_buffer_write_index = 0`
- `static mutex_t telemetry_mutex`  
*Mutex for thread-safe access to the telemetry buffer.*

### 8.105.1 Detailed Description

Implementation of telemetry collection and storage functionality.

Handles collecting, buffering, and persisting telemetry data from various satellite subsystems including power, sensors, and GPS

Definition in file [telemetry\\_manager.cpp](#).

### 8.105.2 Macro Definition Documentation

#### 8.105.2.1 TELEMETRY\_CSV\_PATH

```
#define TELEMETRY_CSV_PATH "/telemetry.csv"
```

Path to the telemetry CSV file on storage media.

Definition at line [28](#) of file [telemetry\\_manager.cpp](#).

#### 8.105.2.2 DEFAULT\_SAMPLE\_INTERVAL\_MS

```
#define DEFAULT_SAMPLE_INTERVAL_MS 2000
```

Default interval between telemetry samples in milliseconds (2 seconds)

Definition at line [33](#) of file [telemetry\\_manager.cpp](#).

#### 8.105.2.3 DEFAULT\_FLUSH\_THRESHOLD

```
#define DEFAULT_FLUSH_THRESHOLD 10
```

Default number of records to collect before flushing to storage.

Definition at line [38](#) of file [telemetry\\_manager.cpp](#).

#### 8.105.2.4 TELEMETRY\_BUFFER\_SIZE

```
#define TELEMETRY_BUFFER_SIZE 20
```

Circular buffer for telemetry records.

Definition at line [121](#) of file [telemetry\\_manager.cpp](#).

### 8.105.3 Variable Documentation

#### 8.105.3.1 powerManager

```
PowerManager powerManager [extern]
```

### 8.105.3.2 systemClock

```
DS3231 systemClock [extern]
```

### 8.105.3.3 nmea\_data

```
NMEAData nmea_data [extern]
```

Definition at line 3 of file [NMEA\\_data.cpp](#).

### 8.105.3.4 sample\_interval\_ms

```
uint32_t sample_interval_ms = DEFAULT_SAMPLE_INTERVAL_MS [static]
```

Current sampling interval in milliseconds.

Definition at line 43 of file [telemetry\\_manager.cpp](#).

### 8.105.3.5 flush\_threshold

```
uint32_t flush_threshold = DEFAULT_FLUSH_THRESHOLD [static]
```

Current flush threshold (number of records that triggers a flush)

Definition at line 48 of file [telemetry\\_manager.cpp](#).

### 8.105.3.6 telemetry\_buffer

```
TelemetryRecord telemetry_buffer[TELEMETRY_BUFFER_SIZE] [static]
```

Definition at line 122 of file [telemetry\\_manager.cpp](#).

### 8.105.3.7 telemetry\_buffer\_count

```
size_t telemetry_buffer_count = 0 [static]
```

Definition at line 123 of file [telemetry\\_manager.cpp](#).

### 8.105.3.8 telemetry\_buffer\_write\_index

```
size_t telemetry_buffer_write_index = 0 [static]
```

Definition at line 124 of file [telemetry\\_manager.cpp](#).

### 8.105.3.9 telemetry\_mutex

```
mutex_t telemetry_mutex [static]
```

Mutex for thread-safe access to the telemetry buffer.

Definition at line 129 of file [telemetry\\_manager.cpp](#).

## 8.106 telemetry\_manager.cpp

[Go to the documentation of this file.](#)

```
00001 #include "telemetry_manager.h"
00002 #include "utils.h"
00003 #include "storage.h"
00004 #include "PowerManager.h"
00005 #include "ISensor.h"
00006 #include "DS3231.h"
00007 #include <deque>
00008 #include <mutex>
00009 #include <iomanip>
00010 #include <sstream>
00011 #include <cstdio>
00012
00019
00020 // External dependencies - these should be documented elsewhere
00021 extern PowerManager powerManager;
00022 extern DS3231 systemClock;
00023 extern NMEAData nmea_data;
00024
00028 #define TELEMETRY_CSV_PATH "/telemetry.csv"
00029
00033 #define DEFAULT_SAMPLE_INTERVAL_MS 2000
00034
00038 #define DEFAULT_FLUSH_THRESHOLD 10
00039
00043 static uint32_t sample_interval_ms = DEFAULT_SAMPLE_INTERVAL_MS;
00044
00048 static uint32_t flush_threshold = DEFAULT_FLUSH_THRESHOLD;
00049
00050
00057 struct TelemetryRecord {
00058     uint32_t timestamp;
00059
00060     // Power data
00061     float battery_voltage;
00062     float system_voltage;
00063     float charge_current_usb;
00064     float charge_current_solar;
00065     float discharge_current;
00066
00067     // Environmental sensor data
00068     float temperature;
00069     float pressure;
00070     float humidity;
00071     float light_level;
00072
00073     // GPS data - key RMC fields
00074     std::string time;
00075     std::string latitude;
00076     std::string lat_dir;
00077     std::string longitude;
00078     std::string lon_dir;
00079     std::string speed;
00080     std::string course;
00081     std::string date;
00082
00083     // GPS data - key GGA fields
00084     std::string fix_quality;
00085     std::string satellites;
00086     std::string altitude;
00087
00088
00089     // Convert record to CSV line
00090     std::string to_csv() const {
00091         std::stringstream ss;
00092         ss << timestamp << ","
00093             << std::fixed << std::setprecision(3)
```

```

00094         << battery_voltage << ","
00095         << system_voltage << ","
00096         << charge_current_usb << ","
00097         << charge_current_solar << ","
00098         << discharge_current << ","
00099         << temperature << ","
00100         << pressure << ","
00101         << humidity << ","
00102         << light_level << ","
00103         // GPS RMC data
00104         << time << ","
00105         << latitude << "," << lat_dir << ","
00106         << longitude << "," << lon_dir << ","
00107         << speed << ","
00108         << course << ","
00109         << date << ","
00110         // GPS GGA data
00111         << fix_quality << ","
00112         << satellites << ","
00113         << altitude;
00114     return ss.str();
00115 }
00116 };
00117
00121 #define TELEMETRY_BUFFER_SIZE 20 // Much smaller than before
00122 static TelemetryRecord telemetry_buffer[TELEMETRY_BUFFER_SIZE];
00123 static size_t telemetry_buffer_count = 0;
00124 static size_t telemetry_buffer_write_index = 0;
00125
00129 static mutex_t telemetry_mutex;
00130
00131 bool telemetry_init() {
00132     mutex_init(&telemetry_mutex);
00133
00134     // Create CSV file with headers if it doesn't exist
00135     if (sd_card_mounted) {
00136         FILE* file = fopen(TELEMETRY_CSV_PATH, "r");
00137         if (!file) {
00138             file = fopen(TELEMETRY_CSV_PATH, "w");
00139             if (file) {
00140                 fprintf(file,
00141                     "timestamp,battery_v,system_v,usb_ma,solar_ma,discharge_ma,temp_c,press_hpa,humidity_pct,light_lux,"
00142                     "gps_time,latitude,lat_dir,longitude,lon_dir,speed_knots,course_deg,date,"
00143                     "fix_quality,satellites,altitude_m\n");
00144                 fclose(file);
00145                 uart_print("Created new telemetry log", VerbosityLevel::INFO);
00146                 return true;
00147             } else {
00148                 uart_print("Failed to create telemetry log", VerbosityLevel::ERROR);
00149                 return false;
00150             }
00151         } else {
00152             fclose(file);
00153             return true;
00154         }
00155     }
00156     uart_print("Telemetry system initialized (storage not available)", VerbosityLevel::WARNING);
00157     return false;
00158 }
00159
00160 bool collect_telemetry() {
00161     SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();
00162
00163     TelemetryRecord record;
00164     record.timestamp = systemClock.get_unix_time();
00165
00166     // Power data
00167     record.battery_voltage = powerManager.get_voltage_battery();
00168     record.system_voltage = powerManager.get_voltage_5v();
00169     record.charge_current_usb = powerManager.get_current_charge_usb();
00170     record.charge_current_solar = powerManager.get_current_charge_solar();
00171     record.discharge_current = powerManager.get_current_draw();
00172
00173     // Sensor data
00174     record.temperature = sensor_wrapper.sensor_read_data(SensorType::ENVIRONMENT,
00175         SensorDataTypeIdentifier::TEMPERATURE);
00176     record.pressure = sensor_wrapper.sensor_read_data(SensorType::ENVIRONMENT,
00177         SensorDataTypeIdentifier::PRESSURE);
00178     record.humidity = sensor_wrapper.sensor_read_data(SensorType::ENVIRONMENT,
00179         SensorDataTypeIdentifier::HUMIDITY);
00180     record.light_level = sensor_wrapper.sensor_read_data(SensorType::LIGHT,
00181         SensorDataTypeIdentifier::LIGHT_LEVEL);
00182
00183     // Get GPS RMC data
00184     std::vector<std::string> rmc_tokens = nmea_data.get_rmc_tokens();

```

```

00181     if (rmc_tokens.size() >= 12) { // RMC has at least 12 fields when complete
00182         record.time = rmc_tokens[1];
00183         record.latitude = rmc_tokens[3];
00184         record.lat_dir = rmc_tokens[4];
00185         record.longitude = rmc_tokens[5];
00186         record.lon_dir = rmc_tokens[6];
00187         record.speed = rmc_tokens[7];
00188         record.course = rmc_tokens[8];
00189         record.date = rmc_tokens[9];
00190     } else {
00191         // Fill with defaults if no GPS data
00192         record.time = "";
00193         record.latitude = "";
00194         record.lat_dir = "";
00195         record.longitude = "";
00196         record.lon_dir = "";
00197         record.speed = "";
00198         record.course = "";
00199         record.date = "";
00200     }
00201
00202     // Get GPS GGA data
00203     std::vector<std::string> gga_tokens = nmea_data.get_gga_tokens();
00204     if (gga_tokens.size() >= 15) { // GGA has 15 fields when complete
00205         record.fix_quality = gga_tokens[6];
00206         record.satellites = gga_tokens[7];
00207         record.altitude = gga_tokens[9];
00208     } else {
00209         // Fill with defaults if no GPS data
00210         record.fix_quality = "";
00211         record.satellites = "";
00212         record.altitude = "";
00213     }
00214
00215     mutex_enter_blocking(&telemetry_mutex);
00216
00217     // Add to circular buffer
00218     telemetry_buffer[telemetry_buffer_write_index] = record;
00219     telemetry_buffer_write_index = (telemetry_buffer_write_index + 1) % TELEMETRY_BUFFER_SIZE;
00220     if (telemetry_buffer_count < TELEMETRY_BUFFER_SIZE) {
00221         telemetry_buffer_count++;
00222     }
00223
00224     mutex_exit(&telemetry_mutex);
00225     return true;
00226 }
00227
00228 bool flush_telemetry() {
00229     if (!sd_card_mounted) {
00230         bool status = fs_init();
00231         if (!status) {
00232             uart_print("Failed to mount storage for telemetry flush", VerbosityLevel::ERROR);
00233             return false;
00234         }
00235     }
00236
00237     mutex_enter_blocking(&telemetry_mutex);
00238
00239     if (telemetry_buffer_count == 0) {
00240         mutex_exit(&telemetry_mutex);
00241         return true; // Nothing to save
00242     }
00243
00244     FILE* file = fopen(TELEMETRY_CSV_PATH, "a");
00245     if (!file) {
00246         uart_print("Failed to open telemetry log for writing", VerbosityLevel::ERROR);
00247         mutex_exit(&telemetry_mutex);
00248         return false;
00249     }
00250
00251     // Calculate start index (for circular buffer)
00252     size_t read_index = 0;
00253     if (telemetry_buffer_count == TELEMETRY_BUFFER_SIZE) {
00254         // Buffer is full, start from oldest entry
00255         read_index = telemetry_buffer_write_index;
00256     }
00257
00258     // Write all records to CSV
00259     for (size_t i = 0; i < telemetry_buffer_count; i++) {
00260         fprintf(file, "%s\n", telemetry_buffer[read_index].to_csv().c_str());
00261         read_index = (read_index + 1) % TELEMETRY_BUFFER_SIZE;
00262     }
00263
00264     // Clear buffer after successful write
00265     telemetry_buffer_count = 0;
00266     telemetry_buffer_write_index = 0;
00267

```

```

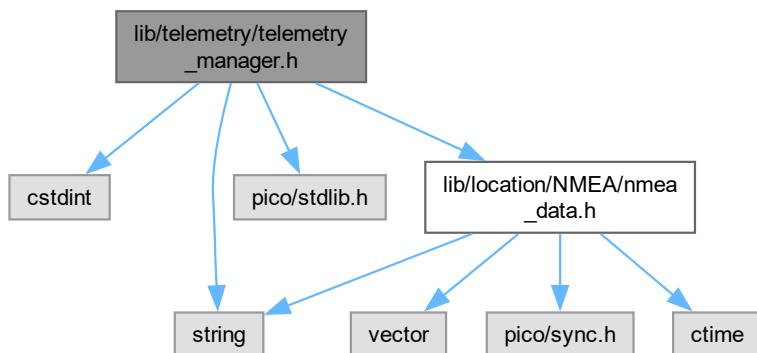
00268     fclose(file);
00269
00270     mutex_exit(&telemetry_mutex);
00271     return true;
00272 }
00273
00274 bool is_telemetry_collection_time(uint32_t current_time, uint32_t& last_collection_time) {
00275     if (current_time - last_collection_time >= sample_interval_ms) {
00276         last_collection_time = current_time;
00277         return true;
00278     }
00279     return false;
00280 }
00281
00282 bool is_telemetry_flush_time(uint32_t& collection_counter) {
00283     if (collection_counter >= flush_threshold) {
00284         collection_counter = 0;
00285         return true;
00286     }
00287     return false;
00288 }
00289
00290 uint32_t get_telemetry_sample_interval() {
00291     return sample_interval_ms;
00292 }
00293
00294 void set_telemetry_sample_interval(uint32_t interval_ms) {
00295     if (interval_ms >= 100) { // Minimum 100ms
00296         sample_interval_ms = interval_ms;
00297     }
00298 }
00299
00300 uint32_t get_telemetry_flush_threshold() {
00301     return flush_threshold;
00302 }
00303
00304 void set_telemetry_flush_threshold(uint32_t records) {
00305     if (records >= 1 && records <= 100) { // Reasonable limits
00306         flush_threshold = records;
00307     }
00308 }

```

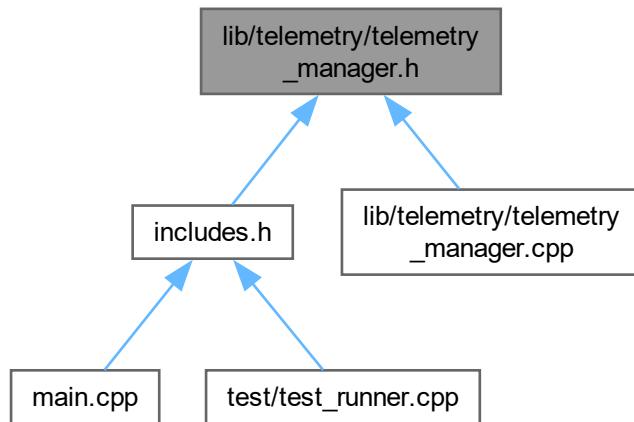
## 8.107 lib/telemetry/telemetry\_manager.h File Reference

System telemetry collection and logging.

```
#include <cstdint>
#include <string>
#include "pico/stdlib.h"
#include "lib/location/NMEA/nmea_data.h"
Include dependency graph for telemetry_manager.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- `bool telemetry_init ()`  
*Initialize the telemetry system.*
- `bool collect_telemetry ()`  
*Collect telemetry data from sensors and power subsystems.*
- `bool flush_telemetry ()`  
*Save buffered telemetry data to storage.*
- `bool is_telemetry_collection_time (uint32_t current_time, uint32_t &last_collection_time)`  
*Check if it's time to collect telemetry based on interval.*
- `bool is_telemetry_flush_time (uint32_t &collection_counter)`  
*Check if it's time to flush telemetry buffer based on count.*
- `uint32_t get_telemetry_sample_interval ()`  
*Get the current sample interval in milliseconds.*
- `void set_telemetry_sample_interval (uint32_t interval_ms)`  
*Set the telemetry sample interval.*
- `uint32_t get_telemetry_flush_threshold ()`  
*Get the number of records before flushing to storage.*
- `void set_telemetry_flush_threshold (uint32_t records)`  
*Set the number of records before flushing to storage.*

### 8.107.1 Detailed Description

System telemetry collection and logging.

This module handles periodic collection and storage of telemetry data from various satellite subsystems including power management, sensors (temperature, pressure, humidity, light), and GPS data.

Telemetry is collected at configurable intervals and stored in a circular buffer before being flushed to persistent storage after a configurable number of records are collected.

Definition in file [telemetry\\_manager.h](#).

## 8.108 telemetry\_manager.h

[Go to the documentation of this file.](#)

```

00001
00015
00016 #ifndef TELEMETRY_MANAGER_H
00017 #define TELEMETRY_MANAGER_H
00018
00019 #include <cstdint>
00020 #include <string>
00021 #include "pico/stdlib.h"
00022 #include "lib/location/NMEA/nmea_data.h"
00023
00030 bool telemetry_init();
00031
00038 bool collect_telemetry();
00039
00046 bool flush_telemetry();
00047
00055 bool is_telemetry_collection_time(uint32_t current_time, uint32_t& last_collection_time);
00056
00063 bool is_telemetry_flush_time(uint32_t& collection_counter);
00064
00069 uint32_t get_telemetry_sample_interval();
00070
00076 void set_telemetry_sample_interval(uint32_t interval_ms);
00077
00082 uint32_t get_telemetry_flush_threshold();
00083
00089 void set_telemetry_flush_threshold(uint32_t records);
00090
00091 #endif // TELEMETRY_MANAGER_H
00092 // End of TelemetryManager group

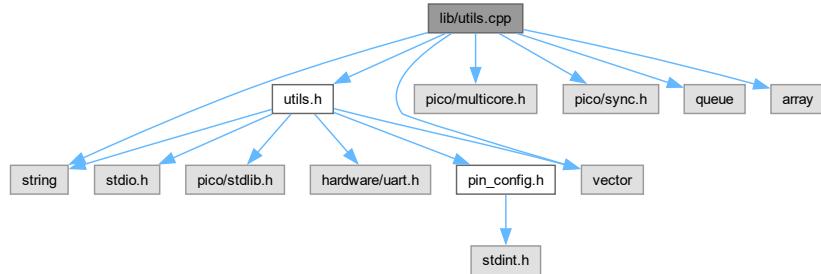
```

## 8.109 lib/utils.cpp File Reference

Implementation of utility functions for the Kubisat firmware.

```
#include "utils.h"
#include "pico/multicore.h"
#include "pico/sync.h"
#include <vector>
#include <queue>
#include <string>
#include <array>
```

Include dependency graph for utils.cpp:



## Functions

- std::string [get\\_level\\_color](#) ([VerbosityLevel](#) level)  
*Gets ANSI color code for verbosity level.*
- std::string [get\\_level\\_prefix](#) ([VerbosityLevel](#) level)  
*Gets text prefix for verbosity level.*
- void [uart\\_print](#) (const std::string &msg, [VerbosityLevel](#) level, bool logToFile, [uart\\_inst\\_t](#) \*uart)  
*Prints a message to the UART with a timestamp and core number.*
- std::string [base64\\_encode](#) (const uint8\_t \*input, size\_t length)
- uint32\_t [calculate\\_checksum](#) (const uint8\_t \*data, size\_t length)

## Variables

- static mutex\_t [uart\\_mutex](#)  
*Mutex for UART access protection.*
- [VerbosityLevel](#) [g\\_uart\\_verbosity](#) = [VerbosityLevel](#)::DEBUG  
*Global verbosity level setting.*
- static const char [base64\\_chars](#) []

### 8.109.1 Detailed Description

Implementation of utility functions for the Kubisat firmware.

Definition in file [utils.cpp](#).

### 8.109.2 Function Documentation

#### 8.109.2.1 [get\\_level\\_color\(\)](#)

```
std::string get_level_color (
    VerbosityLevel level)
```

Gets ANSI color code for verbosity level.

##### Parameters

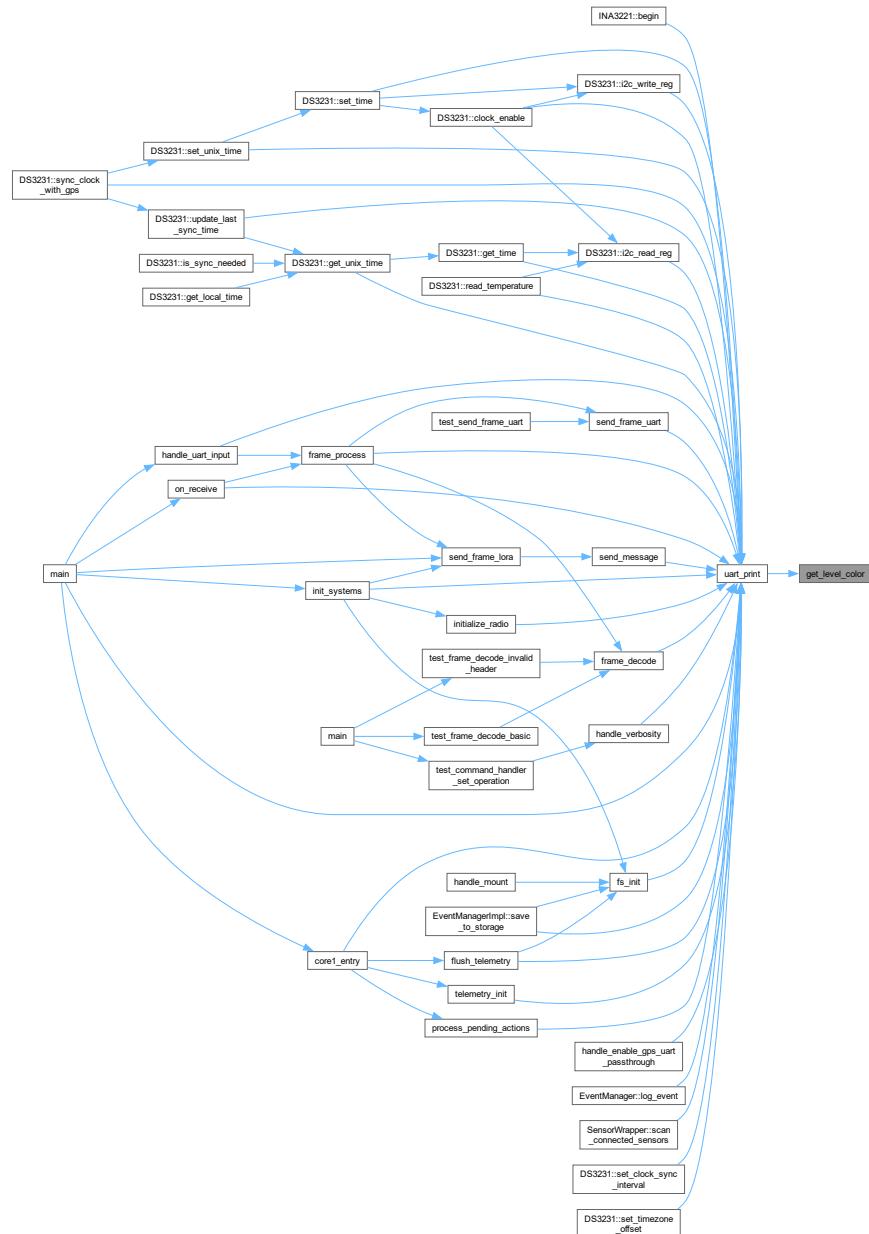
<i>level</i>	The verbosity level
--------------	---------------------

**Returns**

ANSI color escape sequence

Definition at line 29 of file [utils.cpp](#).

Here is the caller graph for this function:

**8.109.2.2 get\_level\_prefix()**

```
std::string get_level_prefix (
    VerbosityLevel level)
```

Gets text prefix for verbosity level.

## Parameters

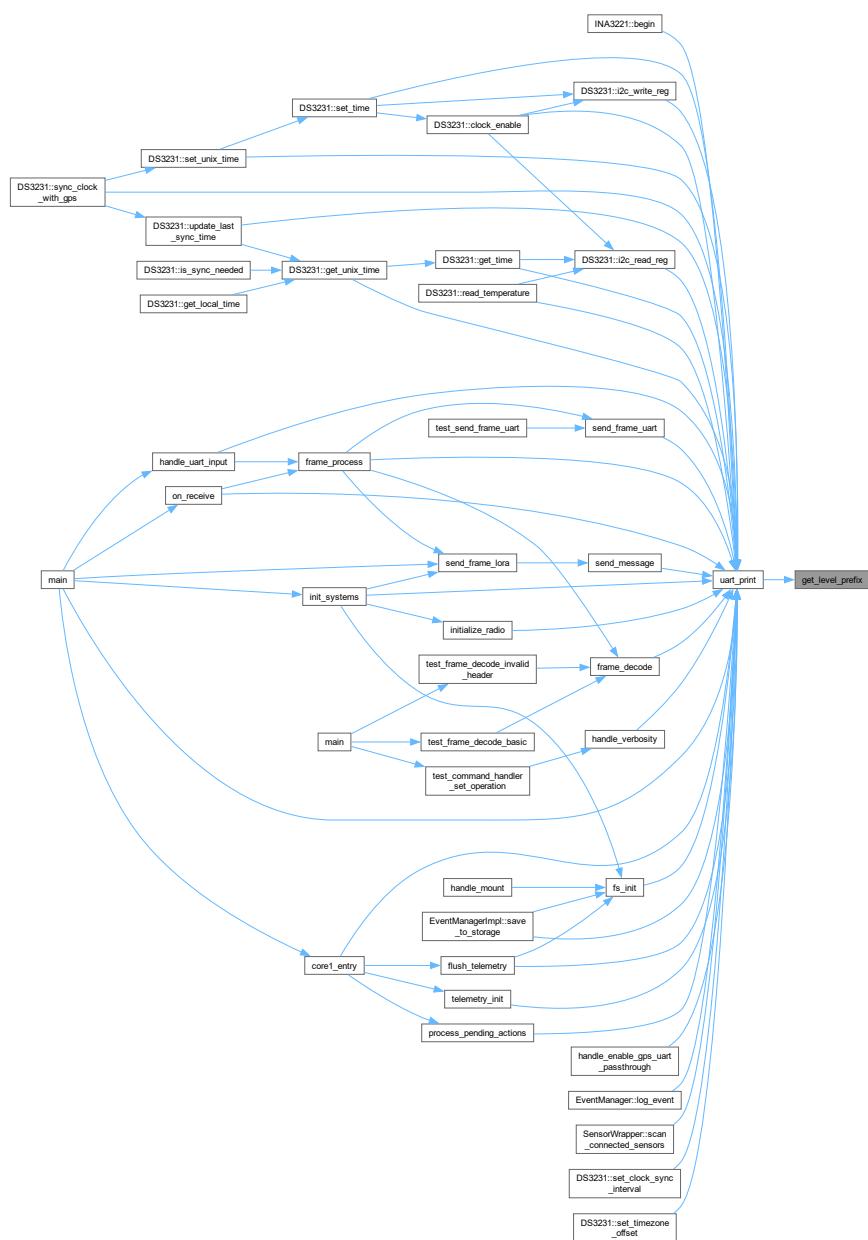
<i>level</i>	The verbosity level
--------------	---------------------

## Returns

Text prefix for the level

Definition at line 46 of file [utils.cpp](#).

Here is the caller graph for this function:



### 8.109.2.3 uart\_print()

```
void uart_print (
    const std::string & msg,
    VerbosityLevel level,
    bool logToFile,
    uart_inst_t * uart)
```

Prints a message to the UART with a timestamp and core number.

Prints a message to UART with timestamp and formatting.

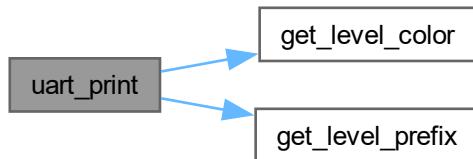
#### Parameters

<i>msg</i>	The message to print.
<i>logToFile</i>	A flag indicating whether to log the message to a file (currently not implemented).
<i>uart</i>	The UART instance to use for printing.

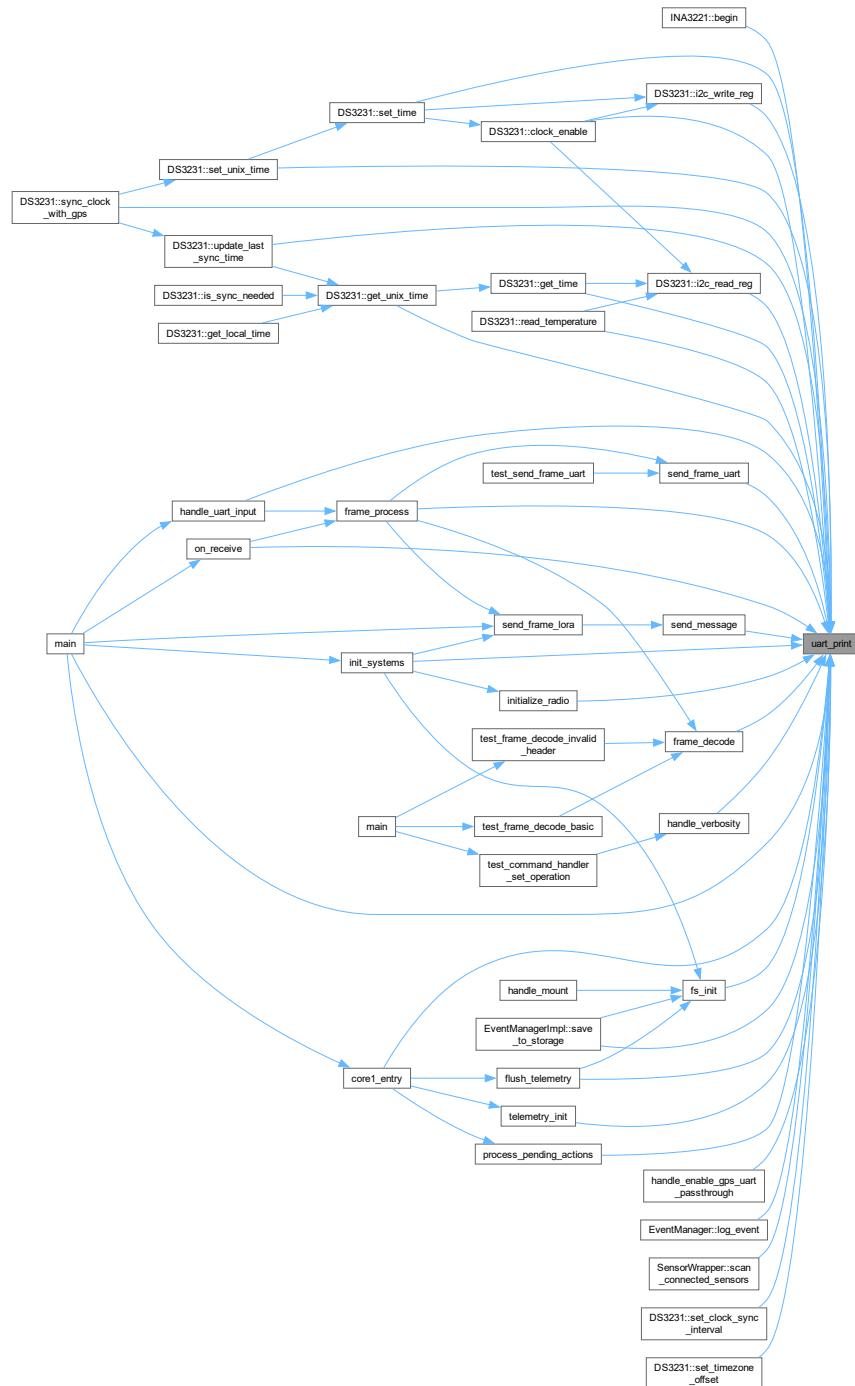
Prints the given message to the specified UART, prepending it with a timestamp and the core number. Uses a mutex to ensure thread-safe access to the UART.

Definition at line 65 of file [utils.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.109.2.4 base64\_encode()

```
std::string base64_encode (
    const uint8_t * input,
    size_t length)
```

Definition at line 95 of file [utils.cpp](#).

### 8.109.2.5 calculate\_checksum()

```
uint32_t calculate_checksum (
    const uint8_t * data,
    size_t length)
```

Definition at line 139 of file [utils.cpp](#).

## 8.109.3 Variable Documentation

### 8.109.3.1 uart\_mutex

```
mutex_t uart_mutex [static]
```

Mutex for UART access protection.

Definition at line 17 of file [utils.cpp](#).

### 8.109.3.2 g\_uart\_verbosity

```
VerbosityLevel g_uart_verbosity = VerbosityLevel::DEBUG
```

Global verbosity level setting.

Global verbosity level setting for UART output.

Definition at line 21 of file [utils.cpp](#).

### 8.109.3.3 base64\_chars

```
const char base64_chars[] [static]
```

#### Initial value:

```
= "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
```

Definition at line 92 of file [utils.cpp](#).

## 8.110 utils.cpp

[Go to the documentation of this file.](#)

```

00001 #include "utils.h"
00002 #include "pico/multicore.h"
00003 #include "pico/sync.h"
00004 #include <vector>
00005 #include <queue>
00006 #include <string>
00007 #include <array>
00008
00009
00014
00015
00017 static mutex_t uart_mutex;
00018
00019
00021 VerbosityLevel g_uart_verbosity = VerbosityLevel::DEBUG;
00022
00023
00029 std::string get_level_color(VerbosityLevel level) {
00030     switch (level) {
00031         case VerbosityLevel::ERROR:    return ANSI_RED;
00032         case VerbosityLevel::WARNING: return ANSI_YELLOW;
00033         case VerbosityLevel::INFO:    return ANSI_GREEN;
00034         case VerbosityLevel::DEBUG:   return ANSI_BLUE;
00035         case VerbosityLevel::EVENT:   return ANSI_CYAN;
00036         default:                      return "";
00037     }
00038 }
00039
00040
00046 std::string get_level_prefix(VerbosityLevel level) {
00047     switch (level) {
00048         case VerbosityLevel::ERROR:    return "ERROR: ";
00049         case VerbosityLevel::WARNING: return "WARNING: ";
00050         case VerbosityLevel::INFO:    return "INFO: ";
00051         case VerbosityLevel::DEBUG:   return "DEBUG: ";
00052         case VerbosityLevel::EVENT:   return "EVENT: ";
00053         default:                      return "";
00054     }
00055 }
00056
00065 void uart_print(const std::string& msg, VerbosityLevel level, bool logToFile, uart_inst_t* uart) {
00066     if (static_cast<int>(level) > static_cast<int>(g_uart_verbosity)) {
00067         return;
00068     }
00069
00070     static bool mutex_initited = false;
00071     if (!mutex_initited) {
00072         mutex_init(&uart_mutex);
00073         mutex_initited = true;
00074     }
00075
00076     uint32_t timestamp = to_ms_since_boot(get_absolute_time());
00077     uint core_num = get_core_num();
00078
00079     std::string color = get_level_color(level);
00080     std::string prefix = get_level_prefix(level);
00081     std::string msg_to_send = "[" + std::to_string(timestamp) + "ms] - Core " +
00082                             std::to_string(core_num) + ":" + +
00083                             color + prefix + ANSI_RESET + msg + "\r\n";
00084
00085     mutex_enter_blocking(&uart_mutex);
00086     uart_puts(uart, msg_to_send.c_str());
00087     mutex_exit(&uart_mutex);
00088 }
00089
00090
00091 // Base64 encoding table
00092 static const char base64_chars[] =
00093     "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
00094
00095 std::string base64_encode(const uint8_t* input, size_t length) {
00096     std::string result;
00097     result.reserve((length + 2) / 3 * 4); // Reserve space for the result
00098
00099     int i = 0;
00100     int j = 0;
00101     uint8_t char_array_3[3];
00102     uint8_t char_array_4[4];
00103
00104     while (length--) {
00105         char_array_3[i++] = *(input++);
00106         if (i == 3) {

```

```

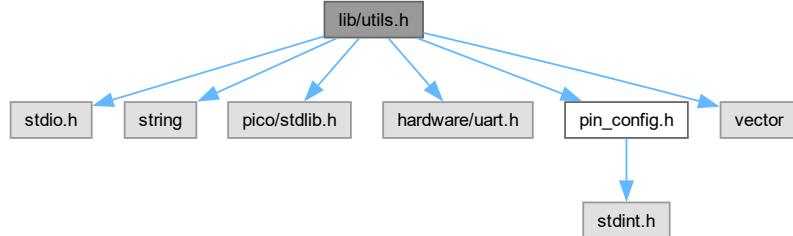
00107     char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
00108     char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0) >> 4);
00109     char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0) >> 6);
00110     char_array_4[3] = char_array_3[2] & 0x3f;
00111
00112     for (i = 0; i < 4; i++)
00113         result += base64_chars[char_array_4[i]];
00114     i = 0;
00115 }
00116 }
00117
00118 // Handle the remaining bytes
00119 if (i) {
00120     for (j = i; j < 3; j++)
00121         char_array_3[j] = '\0';
00122
00123     char_array_4[0] = (char_array_3[0] & 0xfc) >> 2;
00124     char_array_4[1] = ((char_array_3[0] & 0x03) << 4) + ((char_array_3[1] & 0xf0) >> 4);
00125     char_array_4[2] = ((char_array_3[1] & 0x0f) << 2) + ((char_array_3[2] & 0xc0) >> 6);
00126
00127     for (j = 0; j < i + 1; j++)
00128         result += base64_chars[char_array_4[j]];
00129
00130     while (i++ < 3)
00131         result += '='; // Padding
00132 }
00133
00134 return result;
00135 }
00136
00137
00138
00139 uint32_t calculate_checksum(const uint8_t* data, size_t length) {
00140     uint32_t checksum = 0;
00141     for (size_t i = 0; i < length; ++i) {
00142         checksum ^= data[i];
00143     }
00144     return checksum;
00145 }
```

## 8.111 lib/utils.h File Reference

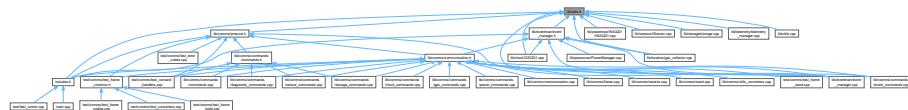
Utility functions and definitions for the Kubisat firmware.

```
#include <stdio.h>
#include <string>
#include "pico/stlplib.h"
#include "hardware/uart.h"
#include "pin_config.h"
#include <vector>
```

Include dependency graph for utils.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define ANSI_RED "\033[31m"`  
*ANSI escape codes for terminal color output.*
- `#define ANSI_GREEN "\033[32m"`
- `#define ANSI_YELLOW "\033[33m"`
- `#define ANSI_BLUE "\033[34m"`
- `#define ANSI_CYAN "\033[36m"`
- `#define ANSI_RESET "\033[0m"`

## Enumerations

- enum class `VerbosityLevel` {
   
    `SILENT` = 0 , `ERROR` = 1 , `WARNING` = 2 , `INFO` = 3 ,
   
    `EVENT` = 4 , `DEBUG` = 5
 }
   
*Verbosity levels for logging system.*

## Functions

- void `uart_print` (const std::string &msg, `VerbosityLevel` level=`VerbosityLevel::INFO`, bool logToFile=false, `uart_inst_t *uart=DEBUG_UART_PORT`)
   
*Prints a message to UART with timestamp and formatting.*
- std::string `base64_encode` (const uint8\_t \*input, size\_t length)
- uint32\_t `calculate_checksum` (const uint8\_t \*data, size\_t length)

## Variables

- `VerbosityLevel g_uart_verbosity`
  
*Global verbosity level setting for UART output.*

### 8.111.1 Detailed Description

Utility functions and definitions for the Kubisat firmware.

Contains UART logging, color definitions, and CRC calculations

Definition in file [utils.h](#).

## 8.111.2 Macro Definition Documentation

### 8.111.2.1 ANSI\_RED

```
#define ANSI_RED "\033[31m"
```

ANSI escape codes for terminal color output.

Definition at line [20](#) of file [utils.h](#).

### 8.111.2.2 ANSI\_GREEN

```
#define ANSI_GREEN "\033[32m"
```

Definition at line [21](#) of file [utils.h](#).

### 8.111.2.3 ANSI\_YELLOW

```
#define ANSI_YELLOW "\033[33m"
```

Definition at line [22](#) of file [utils.h](#).

### 8.111.2.4 ANSI\_BLUE

```
#define ANSI_BLUE "\033[34m"
```

Definition at line [23](#) of file [utils.h](#).

### 8.111.2.5 ANSI\_CYAN

```
#define ANSI_CYAN "\033[36m"
```

Definition at line [24](#) of file [utils.h](#).

### 8.111.2.6 ANSI\_RESET

```
#define ANSI_RESET "\033[0m"
```

Definition at line [25](#) of file [utils.h](#).

## 8.111.3 Enumeration Type Documentation

### 8.111.3.1 VerbosityLevel

```
enum class VerbosityLevel [strong]
```

Verbosity levels for logging system.

**Enumerator**

SILENT	No output
ERROR	Only critical errors
WARNING	Warnings and errors
INFO	Normal operation information
EVENT	Events
DEBUG	Detailed debug information

Definition at line 31 of file [utils.h](#).

## 8.111.4 Function Documentation

### 8.111.4.1 uart\_print()

```
void uart_print (
    const std::string & msg,
    VerbosityLevel level,
    bool logToFile,
    uart_inst_t * uart)
```

Prints a message to UART with timestamp and formatting.

#### Parameters

<i>msg</i>	The message to print
<i>level</i>	Message verbosity level
<i>logToFile</i>	Whether to store the message in log storage
<i>uart</i>	The UART port to use

Prints a message to UART with timestamp and formatting.

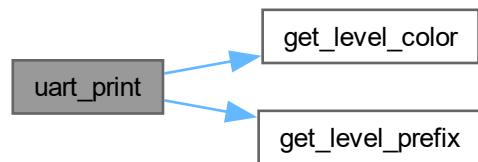
#### Parameters

<i>msg</i>	The message to print.
<i>logToFile</i>	A flag indicating whether to log the message to a file (currently not implemented).
<i>uart</i>	The UART instance to use for printing.

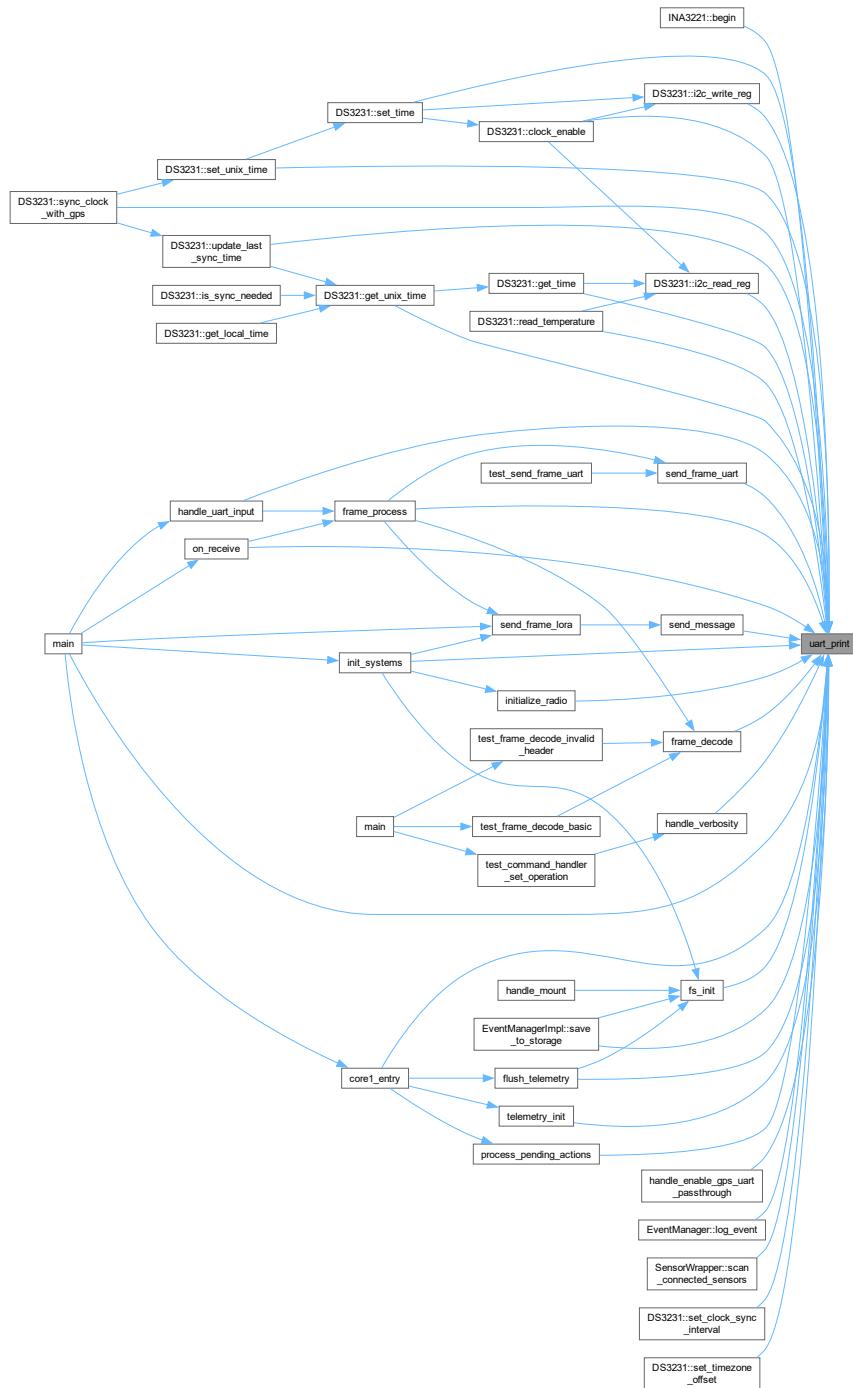
Prints the given message to the specified UART, prepending it with a timestamp and the core number. Uses a mutex to ensure thread-safe access to the UART.

Definition at line 65 of file [utils.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.111.4.2 base64\_encode()

```
std::string base64_encode (
```

```
    const uint8_t * input,
```

```
    size_t length)
```

Definition at line 95 of file [utils.cpp](#).

#### 8.111.4.3 calculate\_checksum()

```
uint32_t calculate_checksum (
    const uint8_t * data,
    size_t length)
```

Definition at line 139 of file [utils.cpp](#).

### 8.111.5 Variable Documentation

#### 8.111.5.1 g\_uart\_verbosity

```
VerbosityLevel g_uart_verbosity [extern]
```

Global verbosity level setting for UART output.

Controls which messages are displayed:

- SILENT (0): No output
- ERROR (1): Only errors
- WARNING (2): Warnings and errors
- INFO (3): Normal operation information
- DEBUG (4): Detailed debug information

#### Note

Can be changed at runtime through the command interface

#### See also

[VerbosityLevel](#)  
[handle\\_verbosity](#)

Global verbosity level setting for UART output.

Definition at line 21 of file [utils.cpp](#).

## 8.112 utils.h

[Go to the documentation of this file.](#)

```
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003
00004 #include <stdio.h>
00005 #include <string>
00006 #include "pico/stl.h"
00007 #include "hardware/uart.h"
00008 #include "pin_config.h"
00009 #include <vector>
00010
00011
00017
00018
00020 #define ANSI_RED      "\033[31m"
00021 #define ANSI_GREEN   "\033[32m"
```

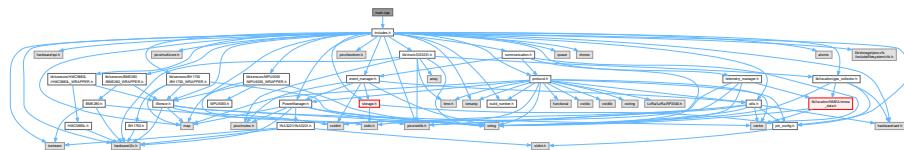
```

00022 #define ANSI_YELLOW "\033[33m"
00023 #define ANSI_BLUE "\033[34m"
00024 #define ANSI_CYAN "\033[36m"
00025 #define ANSI_RESET "\033[0m"
00026
00027
00028 enum class VerbosityLevel {
00029     SILENT = 0,
00030     ERROR = 1,
00031     WARNING = 2,
00032     INFO = 3,
00033     EVENT = 4,
00034     DEBUG = 5
00035 };
00036
00037
00038 };
00039
00040
00053 extern VerbosityLevel g_uart_verbosity;
00054
00055
00063 void uart_print(const std::string& msg,
00064             VerbosityLevel level = VerbosityLevel::INFO,
00065             bool logToFile = false,
00066             uart_inst_t* uart = DEBUG_UART_PORT);
00067
00068
00069
00070 std::string base64_encode(const uint8_t* input, size_t length);
00071
00072 uint32_t calculate_checksum(const uint8_t* data, size_t length);
00073 #endif

```

## 8.113 main.cpp File Reference

#include "includes.h"  
Include dependency graph for main.cpp:



### Macros

- #define LOG\_FILENAME "/log.txt"

### Functions

- void process\_pending\_actions ()
- void core1\_entry ()
- bool init\_systems ()
- int main ()

### Variables

- PowerManager powerManager (MAIN\_I2C\_PORT)
- DS3231 systemClock (MAIN\_I2C\_PORT)
- volatile bool g\_pending\_bootloader\_reset = false
- volatile bool pause\_gps\_collection = false
- char buffer [BUFFER\_SIZE]
- int buffer\_index = 0

### 8.113.1 Macro Definition Documentation

#### 8.113.1.1 LOG\_FILENAME

```
#define LOG_FILENAME "/log.txt"
```

Definition at line 3 of file [main.cpp](#).

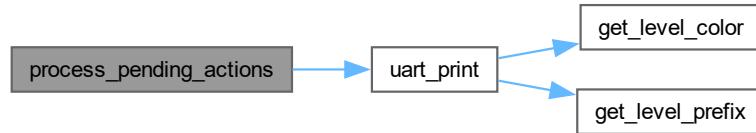
### 8.113.2 Function Documentation

#### 8.113.2.1 process\_pending\_actions()

```
void process_pending_actions ()
```

Definition at line 14 of file [main.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

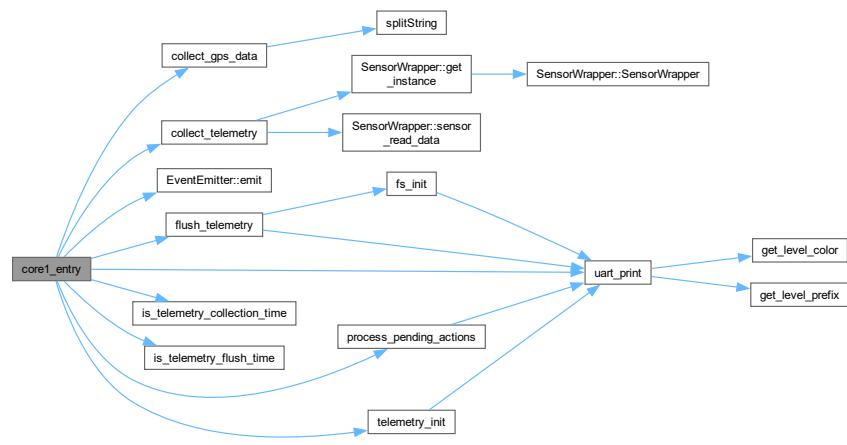


#### 8.113.2.2 core1\_entry()

```
void core1_entry ()
```

Definition at line 23 of file [main.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

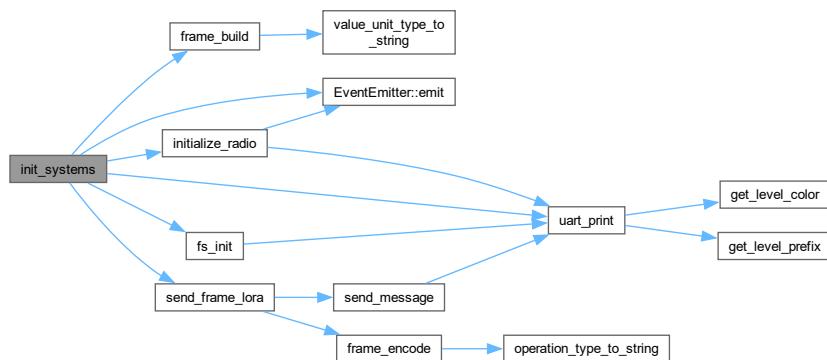


### 8.113.2.3 init\_systems()

```
bool init_systems ()
```

Definition at line 62 of file [main.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

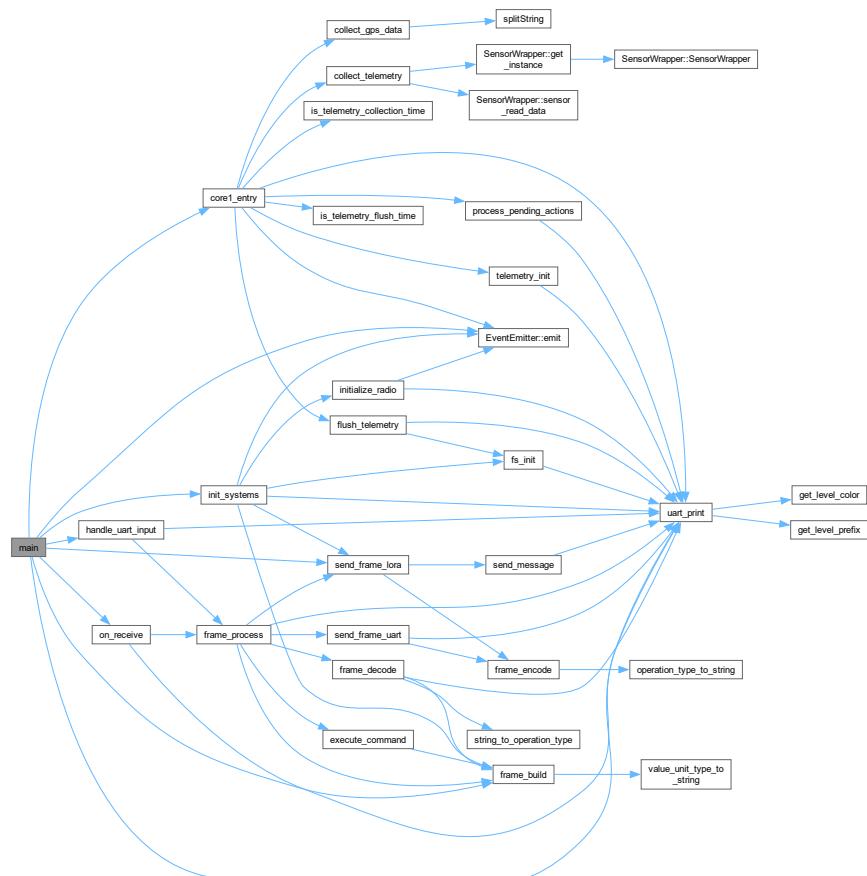


#### 8.113.2.4 main()

```
int main (
    void )
```

Definition at line 136 of file [main.cpp](#).

Here is the call graph for this function:



### 8.113.3 Variable Documentation

#### 8.113.3.1 powerManager

```
PowerManager powerManager(MAIN_I2C_PORT) (
    MAIN_I2C_PORT )
```

#### 8.113.3.2 systemClock

```
DS3231 systemClock(MAIN_I2C_PORT) (
    MAIN_I2C_PORT )
```

#### 8.113.3.3 pause\_gps\_collection

```
volatile bool pause_gps_collection = false
```

Definition at line 8 of file [main.cpp](#).

#### 8.113.3.4 buffer

```
char buffer[BUFFER_SIZE]
```

Definition at line 10 of file [main.cpp](#).

#### 8.113.3.5 buffer\_index

```
int buffer_index = 0
```

Definition at line 11 of file [main.cpp](#).

## 8.114 main.cpp

[Go to the documentation of this file.](#)

```
00001 #include "includes.h"
00002
00003 #define LOG_FILENAME "/log.txt"
00004
00005 PowerManager powerManager(MAIN_I2C_PORT);
00006 DS3231 systemClock(MAIN_I2C_PORT);
00007 volatile bool g_pending_bootloader_reset = false;
00008 volatile bool pause_gps_collection = false;
00009
00010 char buffer[BUFFER_SIZE];
00011 int buffer_index = 0;
00012
00013
00014 void process_pending_actions() {
00015     if (g_pending_bootloader_reset) {
00016         sleep_ms(100);
00017         uart_print("Entering BOOTSEL mode...", VerbosityLevel::WARNING);
00018         reset_usb_boot(0, 0);
00019     }
00020 }
00021
00022
```

```
00023 void core1_entry() {
00024     uart_print("Starting core 1", VerbosityLevel::DEBUG);
00025     EventEmitter::emit(EventGroup::SYSTEM, SystemEvent::CORE1_START);
00026
00027     uint32_t last_clock_check_time = 0;
00028     uint32_t last_telemetry_time = 0;
00029     uint32_t telemetry_collection_counter = 0;
00030
00031     telemetry_init();
00032
00033     while (true) {
00034         collect_gps_data();
00035         process_pending_actions();
00036
00037         uint32_t currentTime = to_ms_since_boot(get_absolute_time());
00038
00039         uint32_t check_interval_ms = systemClock.get_clock_sync_interval() * 60000;
00040         if (currentTime - last_clock_check_time >= check_interval_ms) {
00041             last_clock_check_time = currentTime;
00042
00043             if (systemClock.is_sync_needed()) {
00044                 uart_print("Clock sync interval reached, attempting sync", VerbosityLevel::INFO);
00045                 systemClock.sync_clock_with_gps();
00046             }
00047         }
00048
00049         if (is_telemetry_collection_time(currentTime, last_telemetry_time)) {
00050             collect_telemetry();
00051             telemetry_collection_counter++;
00052
00053             if (is_telemetry_flush_time(telemetry_collection_counter)) {
00054                 flush_telemetry();
00055             }
00056         }
00057
00058         sleep_ms(10);
00059     }
00060 }
00061
00062 bool init_systems() {
00063     stdio_init_all();
00064
00065     uart_init(DEBUG_UART_PORT, DEBUG_UART_BAUD_RATE);
00066     gpio_set_function(DEBUG_UART_TX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_TX_PIN));
00067     gpio_set_function(DEBUG_UART_RX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_RX_PIN));
00068
00069     uart_init(GPS_UART_PORT, GPS_UART_BAUD_RATE);
00070     gpio_set_function(GPS_UART_TX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_TX_PIN));
00071     gpio_set_function(GPS_UART_RX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_RX_PIN));
00072
00073     gpio_init(PICO_DEFAULT_LED_PIN);
00074     gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
00075
00076     i2c_init(MAIN_I2C_PORT, 400 * 1000);
00077     gpio_set_function(MAIN_I2C_SCL_PIN, GPIO_FUNC_I2C);
00078     gpio_set_function(MAIN_I2C_SDA_PIN, GPIO_FUNC_I2C);
00079     gpio_pull_up(MAIN_I2C_SCL_PIN);
00080     gpio_pull_up(MAIN_I2C_SDA_PIN);
00081
00082     gpio_init(GPS_POWER_ENABLE_PIN);
00083     gpio_set_dir(GPS_POWER_ENABLE_PIN, GPIO_OUT);
00084     gpio_put(GPS_POWER_ENABLE_PIN, 1);
00085
00086     EventEmitter::emit(EventGroup::GPS, GPSEvent::POWER_ON);
00087
00088     system("color");
00089
00090     bool radio_init_status = false;
00091     radio_init_status = initialize_radio();
00092
00093     bool sd_init_status = fs_init();
00094     if (sd_init_status) {
00095         FILE *fp = fopen(LOG_FILENAME, "w");
00096         if (fp) {
00097             uart_print("Log file opened.", VerbosityLevel::DEBUG);
00098             int bytes_written = fprintf(fp, "System init started.\n");
00099             uart_print("Written " + std::to_string(bytes_written) + " bytes.", VerbosityLevel::DEBUG);
00100             int close_status = fclose(fp);
00101             uart_print("Close file status: " + std::to_string(close_status), VerbosityLevel::DEBUG);
00102
00103             struct stat file_stat;
00104             if (stat(LOG_FILENAME, &file_stat) == 0) {
00105                 size_t file_size = file_stat.st_size;
00106                 uart_print("File size: " + std::to_string(file_size) + " bytes",
00107                         VerbosityLevel::DEBUG);
00107             } else {
00108                 uart_print("Failed to get file size", VerbosityLevel::ERROR);
00108             }
00109         }
00110     }
00111 }
```

```

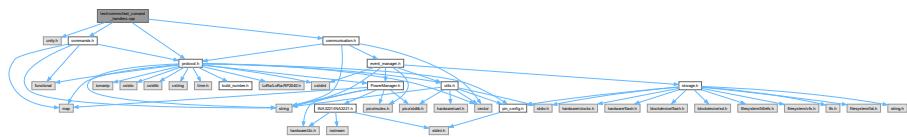
00109         }
00110     uart_print("File path: /" + std::string(LOG_FILENAME), VerbosityLevel::DEBUG);
00111 } else {
00112     uart_print("Failed to open log file for writing.", VerbosityLevel::ERROR);
00113 }
00114 }
00115 }
00116
00117 if (sd_init_status) {
00118     uart_print("SD card init: OK", VerbosityLevel::DEBUG);
00119 } else {
00120     uart_print("SD card init: FAILED", VerbosityLevel::ERROR);
00121 }
00122
00123 if (radio_init_status) {
00124     uart_print("Radio init: OK", VerbosityLevel::DEBUG);
00125 } else {
00126     uart_print("Radio init: FAILED", VerbosityLevel::ERROR);
00127 }
00128
00129 Frame boot = frame_build(OperationType::RES, 0, 0, "HELLO");
00130 send_frame_lora.boot;
00131
00132 return radio_init_status;
00133 }
00134
00135
00136 int main()
00137 {
00138     init_systems();
00139     EventEmitter::emit(EventGroup::SYSTEM, SystemEvent::BOOT);
00140     multicore_launch_core1(core1_entry);
00141
00142     gpio_put(PICO_DEFAULT_LED_PIN, 0);
00143
00144     bool power_manager_init_status = powerManager.initialize();
00145     if (power_manager_init_status)
00146     {
00147         std::map<std::string, std::string> power_config = {
00148             {"operating_mode", "continuous"},
00149             {"averaging_mode", "16"}};
00150     };
00151     powerManager.configure(power_config);
00152 } else {
00153     uart_print("Power manager init error", VerbosityLevel::ERROR);
00154 }
00155
00156 Frame boot = frame_build(OperationType::RES, 0, 0, "START");
00157 send_frame_lora.boot;
00158
00159 std::string boot_string = "System init completed @ " +
00160 std::to_string(to_ms_since_boot(get_absolute_time())) + " ms";
00161 uart.print(boot_string, VerbosityLevel::WARNING);
00162
00163 gpio_put(PICO_DEFAULT_LED_PIN, 1);
00164
00165 while (true)
00166 {
00167     int packet_size = LoRa.parse_packet();
00168     if (packet_size)
00169     {
00170         on_receive(packet_size);
00171     }
00172     handle_uart_input();
00173 }
00174
00175 return 0;
00176 }

```

## 8.115 test/comms/test\_comand\_handlers.cpp File Reference

```
#include "unity.h"
#include "protocol.h"
#include "communication.h"
#include "commands.h"
```

Include dependency graph for test\_comand\_handlers.cpp:



## Functions

- void `send_frame_uart` (const `Frame` &`frame`)
  - void `send_frame_lora` (const `Frame` &`frame`)
  - void `setUp` (void)
  - void `tearDown` (void)
  - void `test_command_handler_get_operation` (void)
  - void `test_command_handler_set_operation` (void)
  - void `test_command_handler_invalid_operation` (void)

## Variables

- static bool `uart_send_called` = false
  - static bool `lora_send_called` = false
  - static Frame `last_frame_sent`

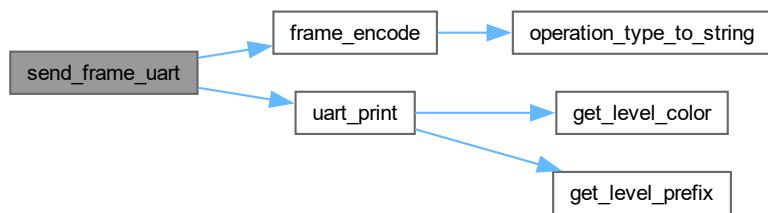
## 8.115.1 Function Documentation

### **8.115.1.1 send\_frame\_uart()**

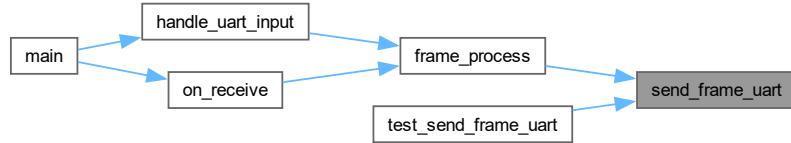
```
void send_frame_uart (
```

Definition at line 11 of file [test\\_comand\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

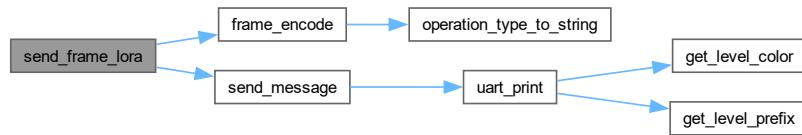


### 8.115.1.2 send\_frame\_lora()

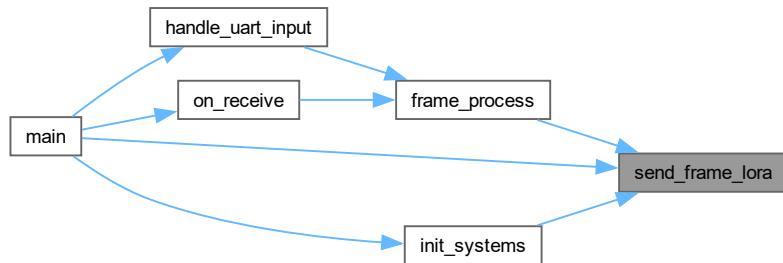
```
void send_frame_lora (
    const Frame & frame)
```

Definition at line 16 of file [test\\_comand\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.115.1.3 setUp()

```
void setUp (
    void )
```

Definition at line 21 of file [test\\_comand\\_handlers.cpp](#).

### 8.115.1.4 tearDown()

```
void tearDown (
    void )
```

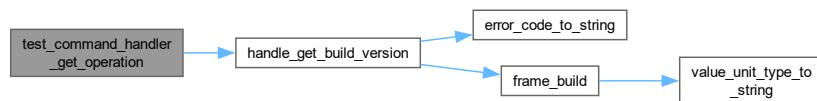
Definition at line 26 of file [test\\_comand\\_handlers.cpp](#).

### 8.115.1.5 test\_command\_handler\_get\_operation()

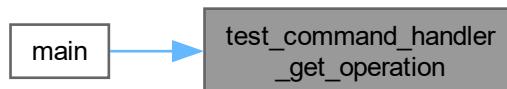
```
void test_command_handler_get_operation (
    void )
```

Definition at line 29 of file [test\\_comand\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

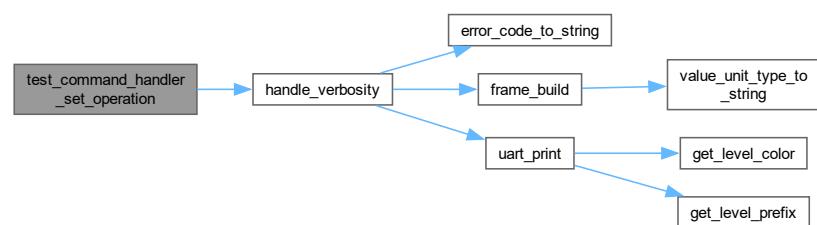


### 8.115.1.6 test\_command\_handler\_set\_operation()

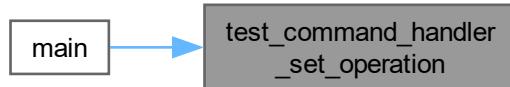
```
void test_command_handler_set_operation (
    void )
```

Definition at line 39 of file [test\\_comand\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

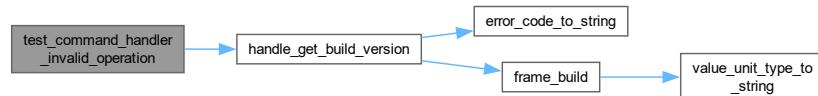


#### 8.115.1.7 `test_command_handler_invalid_operation()`

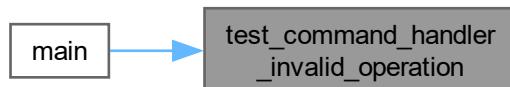
```
void test_command_handler_invalid_operation (
    void )
```

Definition at line 54 of file [test\\_command\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.115.2 Variable Documentation

##### 8.115.2.1 `uart_send_called`

```
bool uart_send_called = false [static]
```

Definition at line 7 of file [test\\_command\\_handlers.cpp](#).

### 8.115.2.2 lora\_send\_called

```
bool lora_send_called = false [static]
```

Definition at line 8 of file [test\\_comand\\_handlers.cpp](#).

### 8.115.2.3 last\_frame\_sent

```
Frame last_frame_sent [static]
```

Definition at line 9 of file [test\\_comand\\_handlers.cpp](#).

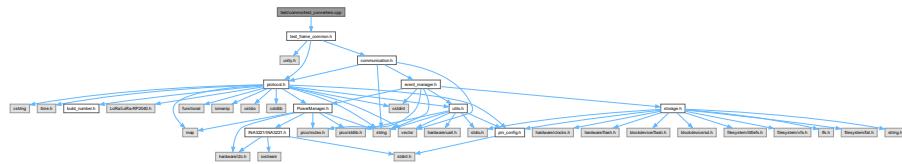
## 8.116 test\_comand\_handlers.cpp

[Go to the documentation of this file.](#)

```
00001 // test/comms/test_command_handlers.cpp
00002 #include "unity.h"
00003 #include "protocol.h"
00004 #include "communication.h"
00005 #include "commands.h"
00006
00007 static bool uart_send_called = false;
00008 static bool lora_send_called = false;
00009 static Frame last_frame_sent;
00010
00011 void send_frame_uart(const Frame& frame) {
00012     uart_send_called = true;
00013     last_frame_sent = frame;
00014 }
00015
00016 void send_frame_lora(const Frame& frame) {
00017     lora_send_called = true;
00018     last_frame_sent = frame;
00019 }
00020
00021 void setUp(void) {
00022     uart_send_called = false;
00023     lora_send_called = false;
00024 }
00025
00026 void tearDown(void) {
00027 }
00028
00029 void test_command_handler_get_operation(void) {
00030     std::vector<Frame> response = handle_get_build_version("", OperationType::GET);
00031
00032     TEST_ASSERT_EQUAL(1, response.size());
00033     TEST_ASSERT_EQUAL(OperationType::VAL, response[0].operationType);
00034     TEST_ASSERT_EQUAL(1, response[0].group);
00035     TEST_ASSERT_EQUAL(1, response[0].command);
00036     TEST_ASSERT_EQUAL(BUILD_NUMBER, std::stoi(response[0].value));
00037 }
00038
00039 void test_command_handler_set_operation(void) {
00040     VerbosityLevel old_level = get_verbosity_level();
00041     std::vector<Frame> response = handle_verbosity("2", OperationType::SET);
00042
00043     TEST_ASSERT_EQUAL(1, response.size());
00044     TEST_ASSERT_EQUAL(OperationType::VAL, response[0].operationType);
00045     TEST_ASSERT_EQUAL(1, response[0].group);
00046     TEST_ASSERT_EQUAL(8, response[0].command);
00047     TEST_ASSERT_EQUAL_STRING("LEVEL SET", response[0].value.c_str());
00048
00049     TEST_ASSERT_EQUAL(VerbosityLevel::WARNING, get_verbosity_level());
00050
00051     set_verbosity_level(old_level);
00052 }
00053
00054 void test_command_handler_invalid_operation(void) {
00055     std::vector<Frame> response = handle_get_build_version("", OperationType::SET);
00056
00057     TEST_ASSERT_EQUAL(1, response.size());
00058     TEST_ASSERT_EQUAL(OperationType::ERR, response[0].operationType);
00059     TEST_ASSERT_EQUAL(1, response[0].group);
00060     TEST_ASSERT_EQUAL(1, response[0].command);
00061 }
```

## 8.117 test/comms/test\_converters.cpp File Reference

```
#include "test_frame_common.h"
Include dependency graph for test_converters.cpp:
```



## Functions

- void `test_operation_type_conversion ()`
  - void `test_value_unit_type_conversion ()`
  - void `test_exception_type_conversion ()`
  - void `test_hex_string_conversion ()`

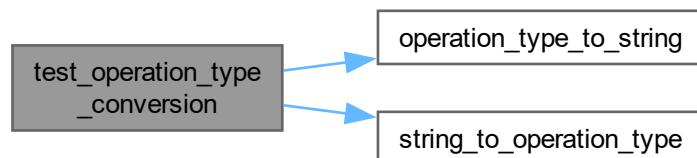
### 8.117.1 Function Documentation

#### **8.117.1.1 test\_operation\_type\_conversion()**

```
void test_operation_type_conversion ( void )
```

Definition at line 4 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

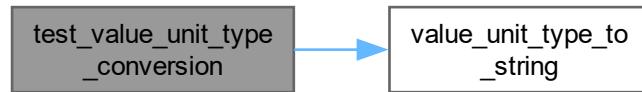


### 8.117.1.2 test\_value\_unit\_type\_conversion()

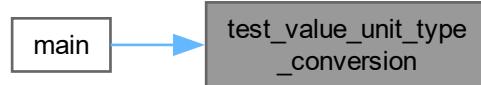
```
void test_value_unit_type_conversion (
    void )
```

Definition at line 13 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

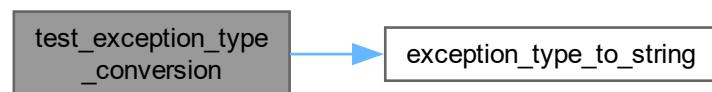


### 8.117.1.3 test\_exception\_type\_conversion()

```
void test_exception_type_conversion (
    void )
```

Definition at line 20 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.117.1.4 `test_hex_string_conversion()`

```
void test_hex_string_conversion (
    void )
```

Definition at line 27 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 8.118 `test_converters.cpp`

[Go to the documentation of this file.](#)

```
00001 // test_frame_converters.cpp
00002 #include "test_frame_common.h"
00003
00004 void test_operation_type_conversion() {
00005     OperationType type = OperationType::GET;
00006     std::string str = operation_type_to_string(type);
```

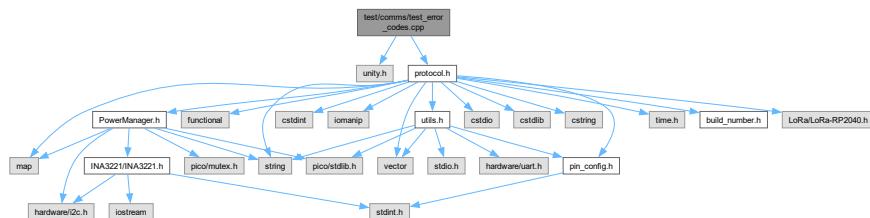
```

00007     OperationType converted = string_to_operation_type(str);
00008
00009     TEST_ASSERT_EQUAL(type, converted);
00010     TEST_ASSERT_EQUAL_STRING("GET", str.c_str());
00011 }
00012
0013 void test_value_unit_type_conversion() {
0014     ValueUnit unit = ValueUnit::VOLT;
0015     std::string str = value_unit_type_to_string(unit);
0016
0017     TEST_ASSERT_EQUAL_STRING("V", str.c_str());
0018 }
0019
0020 void test_exception_type_conversion() {
0021     ExceptionType type = ExceptionType::INVALID_PARAM;
0022     std::string str = exception_type_to_string(type);
0023
0024     TEST_ASSERT_EQUAL_STRING("INVALID PARAM", str.c_str());
0025 }
0026
0027 void test_hex_string_conversion() {
0028     std::string hex = "0A0B0C";
0029     std::vector<uint8_t> bytes = hex_string_to_bytes(hex);
0030
0031     TEST_ASSERT_EQUAL(3, bytes.size());
0032     TEST_ASSERT_EQUAL(0x0A, bytes[0]);
0033     TEST_ASSERT_EQUAL(0x0B, bytes[1]);
0034     TEST_ASSERT_EQUAL(0x0C, bytes[2]);
0035 }

```

## 8.119 test/comms/test\_error\_codes.cpp File Reference

```
#include "unity.h"
#include "protocol.h"
Include dependency graph for test_error_codes.cpp:
```



### Functions

- void [setUp](#) (void)
- void [tearDown](#) (void)
- void [test\\_error\\_code\\_conversion](#) (void)

#### 8.119.1 Function Documentation

##### 8.119.1.1 [setUp\(\)](#)

```
void setUp (
    void )
```

Definition at line 5 of file [test\\_error\\_codes.cpp](#).

### 8.119.1.2 tearDown()

```
void tearDown (
    void )
```

Definition at line 6 of file [test\\_error\\_codes.cpp](#).

### 8.119.1.3 test\_error\_code\_conversion()

```
void test_error_code_conversion (
    void )
```

Definition at line 8 of file [test\\_error\\_codes.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 8.120 test\_error\_codes.cpp

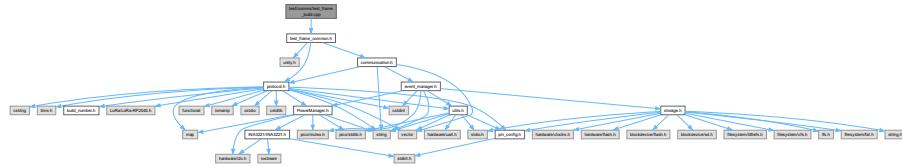
[Go to the documentation of this file.](#)

```

00001 // test/comms/test_error_codes.cpp
00002 #include "unity.h"
00003 #include "protocol.h"
00004
00005 void setUp(void) {}
00006 void tearDown(void) {}
00007
00008 void test_error_code_conversion(void) {
00009     // Test error code to string conversion
0010     TEST_ASSERT_EQUAL_STRING("PARAM_UNNECESSARY",
0011         error_code_to_string(ErrorCode::PARAM_UNNECESSARY).c_str());
0012     TEST_ASSERT_EQUAL_STRING("PARAM_REQUIRED",
0013         error_code_to_string(ErrorCode::PARAM_REQUIRED).c_str());
0014     TEST_ASSERT_EQUAL_STRING("PARAM_INVALID", error_code_to_string(ErrorCode::PARAM_INVALID).c_str());
0015     TEST_ASSERT_EQUAL_STRING("INVALID_OPERATION",
0016         error_code_to_string(ErrorCode::INVALID_OPERATION).c_str());
0017     TEST_ASSERT_EQUAL_STRING("NOT_ALLOWED", error_code_to_string(ErrorCode::NOT_ALLOWED).c_str());
0018     TEST_ASSERT_EQUAL_STRING("INVALID_FORMAT",
0019         error_code_to_string(ErrorCode::INVALID_FORMAT).c_str());
0020     TEST_ASSERT_EQUAL_STRING("INVALID_VALUE", error_code_to_string(ErrorCode::INVALID_VALUE).c_str());
0021     TEST_ASSERT_EQUAL_STRING("UNKNOWN_ERROR", error_code_to_string(ErrorCode::UNKNOWN_ERROR).c_str());
0022 }
```

## 8.121 test/comms/test\_frame\_build.cpp File Reference

```
#include "test_frame_common.h"
Include dependency graph for test_frame_build.cpp:
```



## Functions

- void `test_frame_build_success ()`
  - void `test_frame_build_error ()`
  - void `test_frame_build_info ()`

### 8.121.1 Function Documentation

#### **8.121.1.1 test\_frame\_build\_success()**

```
void test_frame_build_success (
```

Definition at line 4 of file `test_frame_build.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.121.1.2 test\_frame\_build\_error()

```
void test_frame_build_error (
    void )
```

Definition at line 15 of file [test\\_frame\\_build.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.121.1.3 test\_frame\_build\_info()

```
void test_frame_build_info (
    void )
```

Definition at line 24 of file [test\\_frame\\_build.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 8.122 test\_frame\_build.cpp

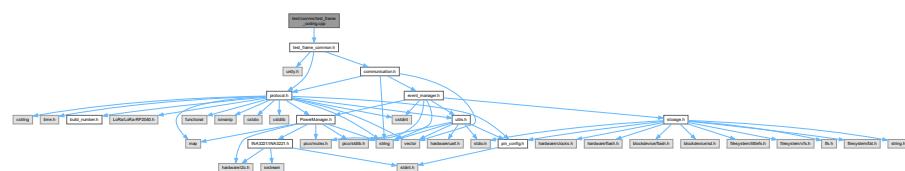
[Go to the documentation of this file.](#)

```
00001 // test_frame_build.cpp
00002 #include "test_frame_common.h"
00003
00004 void test_frame_build_success() {
00005     Frame frame = frame_build(OperationType::VAL, 1, 2, "test_value", ValueUnit::VOLT);
00006
00007     TEST_ASSERT_EQUAL(1, frame.direction);
00008     TEST_ASSERT_EQUAL(OperationType::ANS, frame.operationType);
00009     TEST_ASSERT_EQUAL(1, frame.group);
00010     TEST_ASSERT_EQUAL(2, frame.command);
00011     TEST_ASSERT_EQUAL_STRING("test_value", frame.value.c_str());
00012     TEST_ASSERT_EQUAL_STRING("V", frame.unit.c_str());
00013 }
00014
00015 void test_frame_build_error() {
00016     Frame frame = frame_build(OperationType::ERR, 1, 2, "error_message");
00017
00018     TEST_ASSERT_EQUAL(1, frame.direction);
00019     TEST_ASSERT_EQUAL(OperationType::ERR, frame.operationType);
00020     TEST_ASSERT_EQUAL_STRING("error_message", frame.value.c_str());
00021     TEST_ASSERT_EQUAL_STRING("", frame.unit.c_str());
00022 }
00023
00024 void test_frame_build_info() {
00025     Frame frame = frame_build(OperationType::RES, 1, 2, "info_message");
00026
00027     TEST_ASSERT_EQUAL(1, frame.direction);
00028     TEST_ASSERT_EQUAL(OperationType::INF, frame.operationType);
00029     TEST_ASSERT_EQUAL_STRING("info_message", frame.value.c_str());
00030 }
```

## 8.123 test/comms/test frame coding.cpp File Reference

```
#include "test frame common.h"
```

Include dependency graph for test frame coding.cpp:



## Functions

- void `test_frame_encode_basic()`
  - void `test_frame_decode_basic()`
  - void `test_frame_decode_invalid_header()`

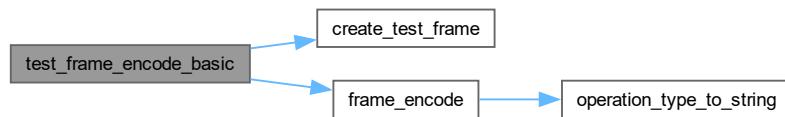
## 8.123.1 Function Documentation

### 8.123.1.1 test\_frame\_encode\_basic()

```
void test_frame_encode_basic (
    void )
```

Definition at line 4 of file [test\\_frame\\_coding.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

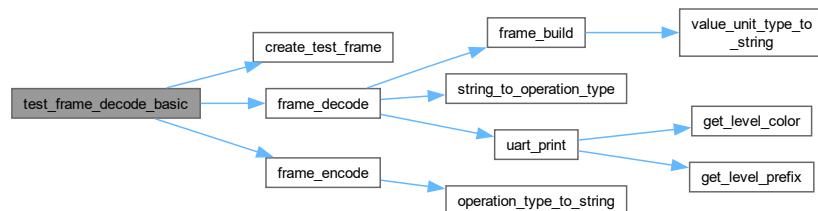


### 8.123.1.2 test\_frame\_decode\_basic()

```
void test_frame_decode_basic (
    void )
```

Definition at line 14 of file [test\\_frame\\_coding.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

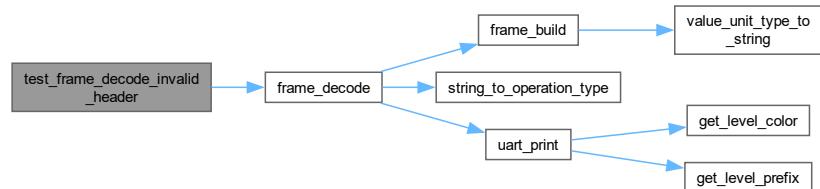


#### 8.123.1.3 test\_frame\_decode\_invalid\_header()

```
void test_frame_decode_invalid_header (
    void )
```

Definition at line 26 of file [test\\_frame\\_coding.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



## 8.124 test\_frame\_coding.cpp

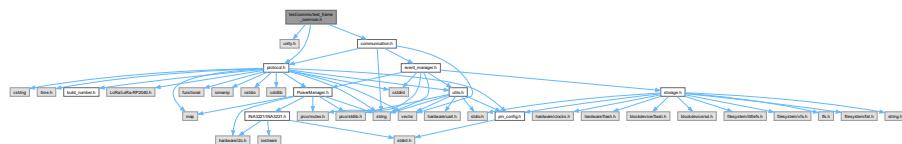
[Go to the documentation of this file.](#)

```
00001 // test_frame_codec.cpp
00002 #include "test_frame_common.h"
00003
00004 void test_frame_encode_basic() {
```

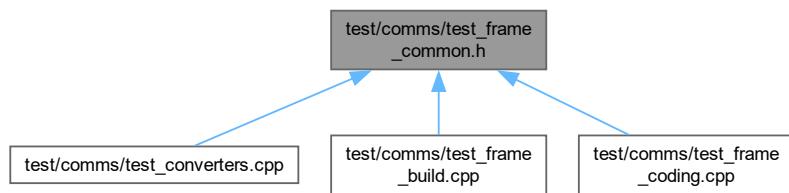
```
00005 Frame frame = create_test_frame();
00006 std::string encoded = frame_encode(frame);
00007
00008 TEST_ASSERT_NOT_EQUAL(0, encoded.length());
00009 TEST_ASSERT_TRUE(encoded.find(FRAME_BEGIN) != std::string::npos);
00010 TEST_ASSERT_TRUE(encoded.find(FRAME_END) != std::string::npos);
00011 TEST_ASSERT_TRUE(encoded.find("test_value") != std::string::npos);
00012 }
00013
00014 void test_frame_decode_basic() {
00015     Frame original = create_test_frame();
00016     std::string encoded = frame_encode(original);
00017     Frame decoded = frame_decode(encoded);
00018
00019     TEST_ASSERT_EQUAL(original.direction, decoded.direction);
00020     TEST_ASSERT_EQUAL(original.group, decoded.group);
00021     TEST_ASSERT_EQUAL(original.command, decoded.command);
00022     TEST_ASSERT_EQUAL_STRING(original.value.c_str(), decoded.value.c_str());
00023     TEST_ASSERT_EQUAL_STRING(original.unit.c_str(), decoded.unit.c_str());
00024 }
00025
00026 void test_frame_decode_invalid_header() {
00027     std::string invalid_frame = "INVALID" + std::string(1, DELIMITER) + "rest_of_frame";
00028     bool exceptionThrown = false;
00029
00030     try {
00031         Frame decoded = frame_decode(invalid_frame);
00032     } catch (const std::runtime_error& e) {
00033         exceptionThrown = true;
00034     } catch (...) {
00035         // Catch any other exceptions to avoid crashing the test
00036     }
00037
00038     TEST_ASSERT_TRUE(exceptionThrown);
00039 }
```

## 8.125 test/comms/test\_frame\_common.h File Reference

```
#include "unity.h"
#include "protocol.h"
#include "communication.h"
Include dependency graph for test_frame_common.h:
```



This graph shows which files directly or indirectly include this file:



## Functions

- Frame create\_test\_frame ()

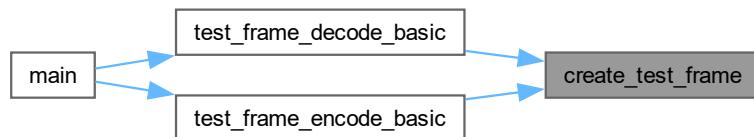
### 8.125.1 Function Documentation

#### 8.125.1.1 create\_test\_frame()

Frame create\_test\_frame ()

Definition at line 10 of file [test\\_frame\\_common.h](#).

Here is the caller graph for this function:



## 8.126 test\_frame\_common.h

[Go to the documentation of this file.](#)

```
00001 // test_frame_common.h
00002 #ifndef TEST_FRAME_COMMON_H
00003 #define TEST_FRAME_COMMON_H
00004
00005 #include "unity.h"
00006 #include "protocol.h"
00007 #include "communication.h"
00008
00009 // Helper function to create a test frame
00010 Frame create_test_frame() {
00011     Frame frame;
00012     frame.header = FRAME_BEGIN;
00013     frame.direction = 1;
00014     frame.operationType = OperationType::GET;
00015     frame.group = 1;
00016     frame.command = 2;
00017     frame.value = "test_value";
00018     frame.unit = "V";
00019     frame.footer = FRAME_END;
00020     return frame;
00021 }
00022
00023 #endif
```

## 8.127 test/comms/test\_frame\_send.cpp File Reference

```
#include "unity.h"
#include "communication.h"
#include "../mocks/hardwareMocks.h"
Include dependency graph for test_frame_send.cpp:
```



### Functions

- void [setUp](#) (void)
- void [tearDown](#) (void)
- void [test\\_send\\_frame\\_uart](#) (void)

#### 8.127.1 Function Documentation

##### 8.127.1.1 [setUp\(\)](#)

```
void setUp (
    void )
```

Definition at line [6](#) of file [test\\_frame\\_send.cpp](#).

##### 8.127.1.2 [tearDown\(\)](#)

```
void tearDown (
    void )
```

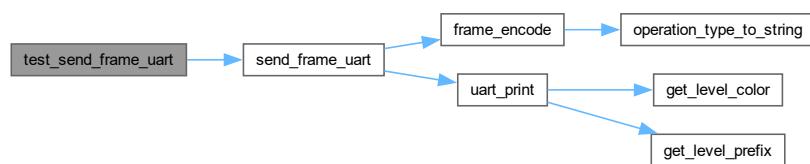
Definition at line [12](#) of file [test\\_frame\\_send.cpp](#).

##### 8.127.1.3 [test\\_send\\_frame\\_uart\(\)](#)

```
void test_send_frame_uart (
    void )
```

Definition at line [17](#) of file [test\\_frame\\_send.cpp](#).

Here is the call graph for this function:



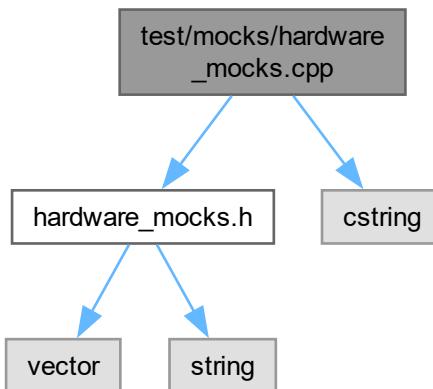
## 8.128 test\_frame\_send.cpp

Go to the documentation of this file.

```
00001 // test/comms/test_frame_send.cpp
00002 #include "unity.h"
00003 #include "communication.h"
00004 #include "../mocks/hardware_mocks.h"
00005
00006 void setUp(void) {
00007     // Enable mocks before each test
00008     mock_uart_enabled = true;
00009     uart_output_buffer.clear();
00010 }
00011
00012 void tearDown(void) {
00013     // Disable mocks after each test
00014     mock_uart_enabled = false;
00015 }
00016
00017 void test_send_frame_uart(void) {
00018     // Create a test frame
00019     Frame test_frame = {
00020         .operationType = OperationType::VAL,
00021         .group = 1,
00022         .command = 2,
00023         .value = "TEST_VALUE"
00024     };
00025
00026     // Call function under test
00027     send_frame_uart(test_frame);
00028
00029     // Verify output using mocks
00030     TEST_ASSERT_EQUAL(1, uart_output_buffer.size());
00031     TEST_ASSERT_TRUE(uart_output_buffer[0].find("KBST;0;VAL;1;2;TEST_VALUE;") != std::string::npos);
00032 }
```

## 8.129 test/mockshardware\_mocks.cpp File Reference

```
#include "hardware_mocks.h"
#include <cstring>
Include dependency graph for hardware_mocks.cpp:
```



## Functions

- void `mock_uart_puts` (uart\_inst\_t \*uart, const char \*str)
- void `mock_uart_init` (uart\_inst\_t \*uart, uint baudrate)
- void `mock_spi_write_blocking` (spi\_inst\_t \*spi, const uint8\_t \*src, size\_t len)
- int `mock_spi_read_blocking` (spi\_inst\_t \*spi, uint8\_t tx\_data, uint8\_t \*dst, size\_t len)

## Variables

- bool `mock_uart_enabled` = false
- std::vector< std::string > `uart_output_buffer`
- bool `mock_spi_enabled` = false
- std::vector< uint8\_t > `spi_output_buffer`

## 8.129.1 Function Documentation

### 8.129.1.1 `mock_uart_puts()`

```
void mock_uart_puts (
    uart_inst_t * uart,
    const char * str)
```

Definition at line 9 of file [hardwareMocks.cpp](#).

### 8.129.1.2 `mock_uart_init()`

```
void mock_uart_init (
    uart_inst_t * uart,
    uint baudrate)
```

Definition at line 17 of file [hardwareMocks.cpp](#).

### 8.129.1.3 `mock_spi_write_blocking()`

```
void mock_spi_write_blocking (
    spi_inst_t * spi,
    const uint8_t * src,
    size_t len)
```

Definition at line 27 of file [hardwareMocks.cpp](#).

### 8.129.1.4 `mock_spi_read_blocking()`

```
int mock_spi_read_blocking (
    spi_inst_t * spi,
    uint8_t tx_data,
    uint8_t * dst,
    size_t len)
```

Definition at line 35 of file [hardwareMocks.cpp](#).

## 8.129.2 Variable Documentation

### 8.129.2.1 mock\_uart\_enabled

```
bool mock_uart_enabled = false
```

Definition at line 6 of file [hardware\\_mocks.cpp](#).

### 8.129.2.2 uart\_output\_buffer

```
std::vector<std::string> uart_output_buffer
```

Definition at line 7 of file [hardware\\_mocks.cpp](#).

### 8.129.2.3 mock\_spi\_enabled

```
bool mock_spi_enabled = false
```

Definition at line 24 of file [hardware\\_mocks.cpp](#).

### 8.129.2.4 spi\_output\_buffer

```
std::vector<uint8_t> spi_output_buffer
```

Definition at line 25 of file [hardware\\_mocks.cpp](#).

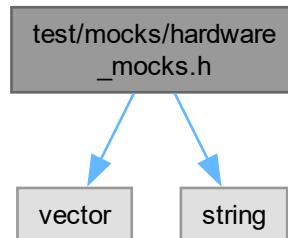
## 8.130 hardware\_mocks.cpp

[Go to the documentation of this file.](#)

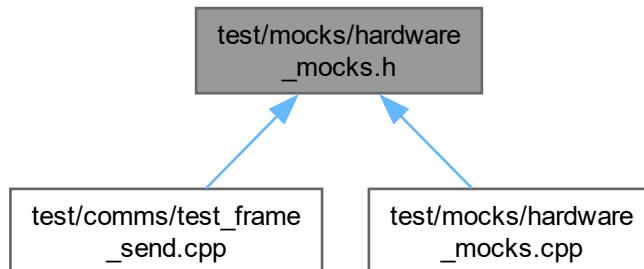
```
00001 // test/mock/hardware_mocks.cpp
00002 #include "hardware_mocks.h"
00003 #include <cstring>
00004
00005 // UART mocks
00006 bool mock_uart_enabled = false;
00007 std::vector<std::string> uart_output_buffer;
00008
00009 void mock_uart_puts(uart_inst_t* uart, const char* str) {
00010     if (mock_uart_enabled) {
00011         uart_output_buffer.push_back(std::string(str));
00012     } else {
00013         uart_puts(uart, str);
00014     }
00015 }
00016
00017 void mock_uart_init(uart_inst_t* uart, uint baudrate) {
00018     if (!mock_uart_enabled) {
00019         uart_init(uart, baudrate);
00020     }
00021 }
00022
00023 // SPI mocks
00024 bool mock_spi_enabled = false;
00025 std::vector<uint8_t> spi_output_buffer;
00026
00027 void mock_spi_write_blocking(spi_inst_t* spi, const uint8_t* src, size_t len) {
00028     if (mock_spi_enabled) {
00029         spi_output_buffer.insert(spi_output_buffer.end(), src, src + len);
00030     } else {
00031         spi_write_blocking(spi, src, len);
00032     }
00033 }
00034
00035 int mock_spi_read_blocking(spi_inst_t* spi, uint8_t tx_data, uint8_t* dst, size_t len) {
00036     if (mock_spi_enabled) {
00037         // Mock implementation that fills dst with test data
00038         memset(dst, tx_data, len);
00039         return len;
00040     } else {
00041         return spi_read_blocking(spi, tx_data, dst, len);
00042     }
00043 }
```

## 8.131 test/mocks/hardware\_mocksh File Reference

```
#include <vector>
#include <string>
Include dependency graph for hardware_mocksh:
```



This graph shows which files directly or indirectly include this file:



### Functions

- void [mock\\_uart\\_puts](#) (uart\_inst\_t \*uart, const char \*str)
- void [mock\\_uart\\_init](#) (uart\_inst\_t \*uart, uint baudrate)
- void [mock\\_spi\\_write\\_blocking](#) (spi\_inst\_t \*spi, const uint8\_t \*src, size\_t len)
- int [mock\\_spi\\_read\\_blocking](#) (spi\_inst\_t \*spi, uint8\_t tx\_data, uint8\_t \*dst, size\_t len)

### Variables

- bool [mock\\_uart\\_enabled](#)
- std::vector< std::string > [uart\\_output\\_buffer](#)
- bool [mock\\_spi\\_enabled](#)
- std::vector< uint8\_t > [spi\\_output\\_buffer](#)

## 8.131.1 Function Documentation

### 8.131.1.1 mock\_uart\_puts()

```
void mock_uart_puts (
    uart_inst_t * uart,
    const char * str)
```

Definition at line 9 of file [hardware\\_mock.cpp](#).

### 8.131.1.2 mock\_uart\_init()

```
void mock_uart_init (
    uart_inst_t * uart,
    uint baudrate)
```

Definition at line 17 of file [hardware\\_mock.cpp](#).

### 8.131.1.3 mock\_spi\_write\_blocking()

```
void mock_spi_write_blocking (
    spi_inst_t * spi,
    const uint8_t * src,
    size_t len)
```

Definition at line 27 of file [hardware\\_mock.cpp](#).

### 8.131.1.4 mock\_spi\_read\_blocking()

```
int mock_spi_read_blocking (
    spi_inst_t * spi,
    uint8_t tx_data,
    uint8_t * dst,
    size_t len)
```

Definition at line 35 of file [hardware\\_mock.cpp](#).

## 8.131.2 Variable Documentation

### 8.131.2.1 mock\_uart\_enabled

```
bool mock_uart_enabled [extern]
```

Definition at line 6 of file [hardware\\_mock.cpp](#).

### 8.131.2.2 uart\_output\_buffer

```
std::vector<std::string> uart_output_buffer [extern]
```

Definition at line 7 of file [hardware\\_mock.cpp](#).

### 8.131.2.3 mock\_spi\_enabled

```
bool mock_spi_enabled [extern]
```

Definition at line 24 of file [hardwareMocks.cpp](#).

### 8.131.2.4 spi\_output\_buffer

```
std::vector<uint8_t> spi_output_buffer [extern]
```

Definition at line 25 of file [hardwareMocks.cpp](#).

## 8.132 hardwareMocks.h

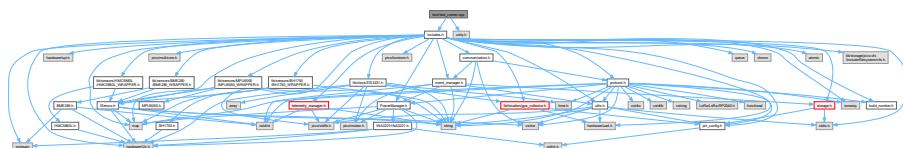
[Go to the documentation of this file.](#)

```
00001 // test/mock/hardwareMocks.h
00002 #ifndef HARDWARE_MOCKS_H
00003 #define HARDWARE_MOCKS_H
00004
00005 #include <vector>
00006 #include <string>
00007
00008 // UART mocks
00009 extern bool mock_uart_enabled;
00010 extern std::vector<std::string> uart_output_buffer;
00011
00012 void mock_uart_puts(uart_inst_t* uart, const char* str);
00013 void mock_uart_init(uart_inst_t* uart, uint baudrate);
00014
00015 // SPI mocks
00016 extern bool mock_spi_enabled;
00017 extern std::vector<uint8_t> spi_output_buffer;
00018
00019 void mock_spi_write_blocking(spi_inst_t* spi, const uint8_t* src, size_t len);
00020 int mock_spi_read_blocking(spi_inst_t* spi, uint8_t tx_data, uint8_t* dst, size_t len);
00021
00022 #endif // HARDWARE_MOCKS_H
```

## 8.133 test/test\_runner.cpp File Reference

```
#include "includes.h"
#include "unity.h"
```

Include dependency graph for `test_runner.cpp`:



## Functions

- void `test_frame_encode_basic` (void)
- void `test_frame_decode_basic` (void)
- void `test_frame_decode_invalid_header` (void)
- void `test_frame_build_success` (void)
- void `test_frame_build_error` (void)
- void `test_frame_build_info` (void)
- void `test_operation_type_conversion` (void)
- void `test_value_unit_type_conversion` (void)
- void `test_exception_type_conversion` (void)
- void `test_hex_string_conversion` (void)
- void `test_command_handler_get_operation` (void)
- void `test_command_handler_set_operation` (void)
- void `test_command_handler_invalid_operation` (void)
- void `test_error_code_conversion` (void)
- int `main` (void)

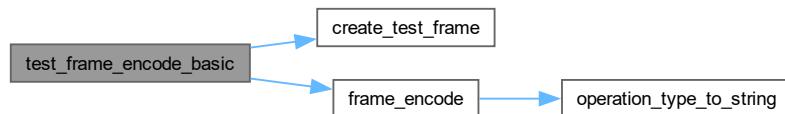
### 8.133.1 Function Documentation

#### 8.133.1.1 `test_frame_encode_basic()`

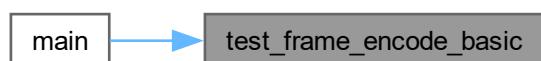
```
void test_frame_encode_basic (
    void ) [extern]
```

Definition at line 4 of file `test_frame_coding.cpp`.

Here is the call graph for this function:



Here is the caller graph for this function:

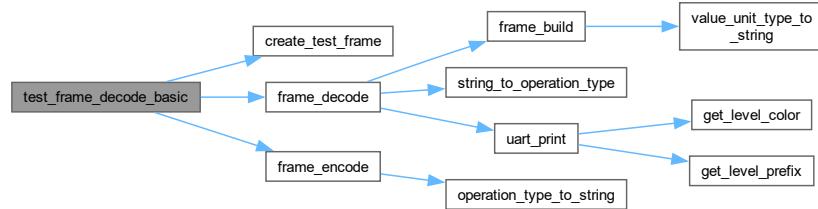


### 8.133.1.2 test\_frame\_decode\_basic()

```
void test_frame_decode_basic (
    void )  [extern]
```

Definition at line 14 of file [test\\_frame\\_coding.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

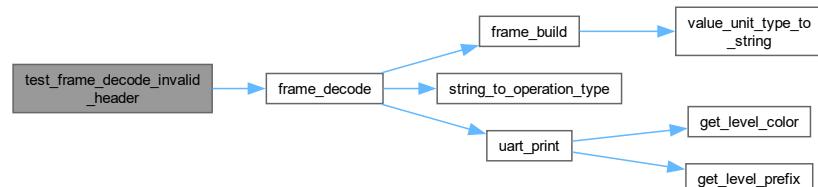


### 8.133.1.3 test\_frame\_decode\_invalid\_header()

```
void test_frame_decode_invalid_header (
    void )  [extern]
```

Definition at line 26 of file [test\\_frame\\_coding.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

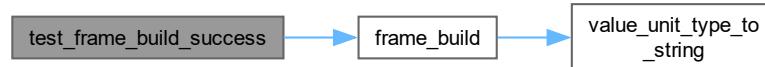


#### 8.133.1.4 test\_frame\_build\_success()

```
void test_frame_build_success (
    void ) [extern]
```

Definition at line 4 of file [test\\_frame\\_build.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

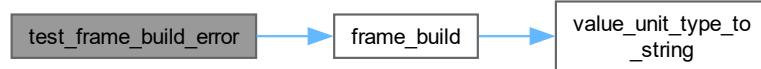


#### 8.133.1.5 test\_frame\_build\_error()

```
void test_frame_build_error (
    void ) [extern]
```

Definition at line 15 of file [test\\_frame\\_build.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



#### 8.133.1.6 test\_frame\_build\_info()

```
void test_frame_build_info (
    void ) [extern]
```

Definition at line 24 of file [test\\_frame\\_build.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

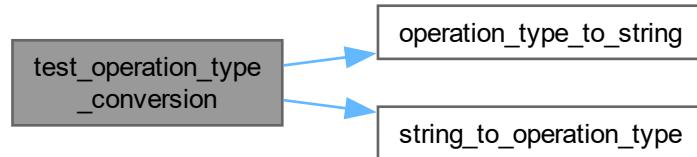


### 8.133.1.7 test\_operation\_type\_conversion()

```
void test_operation_type_conversion (
    void ) [extern]
```

Definition at line 4 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

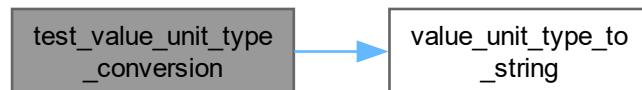


### 8.133.1.8 test\_value\_unit\_type\_conversion()

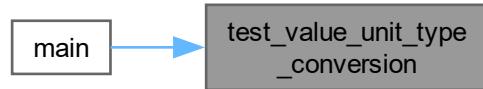
```
void test_value_unit_type_conversion (
    void ) [extern]
```

Definition at line 13 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

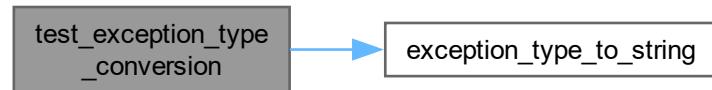


#### 8.133.1.9 `test_exception_type_conversion()`

```
void test_exception_type_conversion (
    void ) [extern]
```

Definition at line 20 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.133.1.10 test\_hex\_string\_conversion()

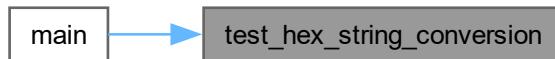
```
void test_hex_string_conversion (
    void ) [extern]
```

Definition at line 27 of file [test\\_converters.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

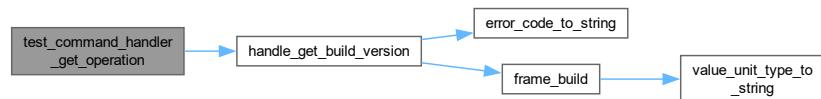


### 8.133.1.11 test\_command\_handler\_get\_operation()

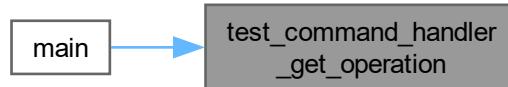
```
void test_command_handler_get_operation (
    void ) [extern]
```

Definition at line 29 of file [test\\_command\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

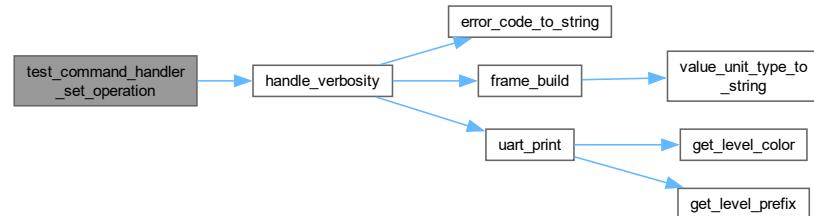


#### 8.133.1.12 `test_command_handler_set_operation()`

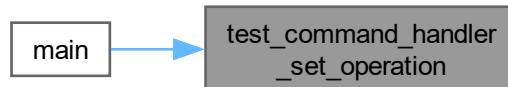
```
void test_command_handler_set_operation (
    void )  [extern]
```

Definition at line 39 of file [test\\_command\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

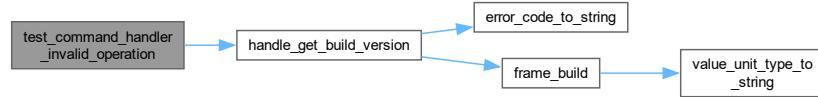


### 8.133.1.13 test\_command\_handler\_invalid\_operation()

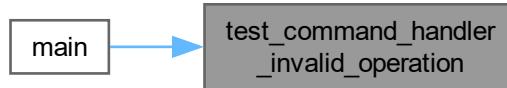
```
void test_command_handler_invalid_operation (
    void )  [extern]
```

Definition at line 54 of file [test\\_command\\_handlers.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.133.1.14 test\_error\_code\_conversion()

```
void test_error_code_conversion (
    void )  [extern]
```

Definition at line 8 of file [test\\_error\\_codes.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

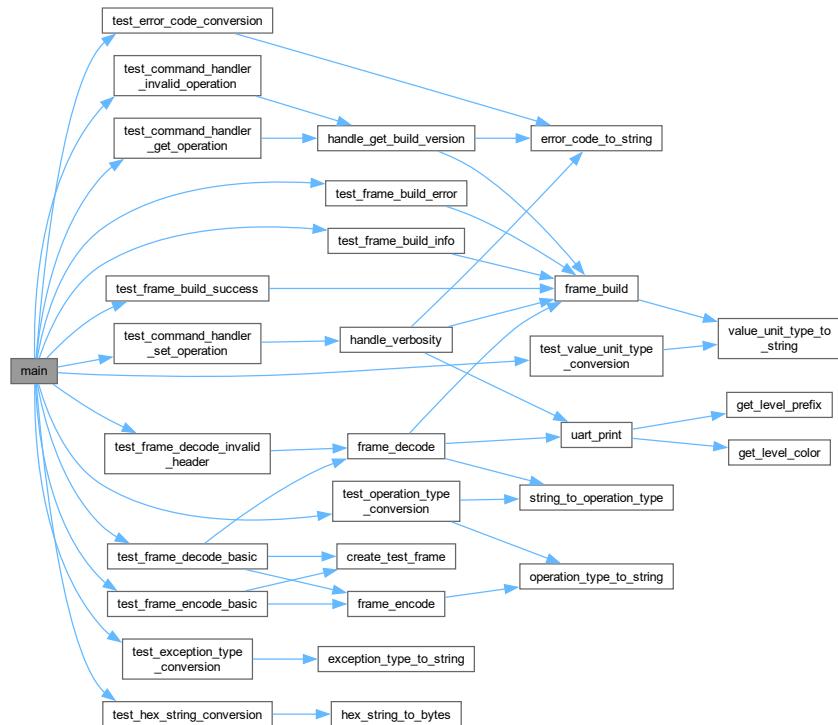


### 8.133.1.15 main()

```
int main (
    void )
```

Definition at line 26 of file [test\\_runner.cpp](#).

Here is the call graph for this function:



## 8.134 test\_runner.cpp

[Go to the documentation of this file.](#)

```
00001 // test/test_runner.cpp
```

```
00002 #include "includes.h"
00003 #include "unity.h"
00004
00005 // External test function declarations
00006 // Pure software tests (no hardware dependencies)
00007 extern void test_frame_encode_basic(void);
00008 extern void test_frame_decode_basic(void);
00009 extern void test_frame_decode_invalid_header(void);
00010 extern void test_frame_build_success(void);
00011 extern void test_frame_build_error(void);
00012 extern void test_frame_build_info(void);
00013 extern void test_operation_type_conversion(void);
00014 extern void test_value_unit_type_conversion(void);
00015 extern void test_exception_type_conversion(void);
00016 extern void test_hex_string_conversion(void);
00017
00018 // Command handler tests
00019 extern void test_command_handler_get_operation(void);
00020 extern void test_command_handler_set_operation(void);
00021 extern void test_command_handler_invalid_operation(void);
00022
00023 // Error code tests
00024 extern void test_error_code_conversion(void);
00025
00026 int main(void) {
00027     stdio_init_all();
00028     uart_init(uart0, 115200);
00029     gpio_set_function(0, GPIO_FUNC_UART);
00030     gpio_set_function(1, GPIO_FUNC_UART);
00031
00032     UNITY_BEGIN();
00033     uart_puts(uart0, "begin unity tests\n");
00034
00035     // Frame codec tests (pure software)
00036     uart_puts(uart0, "begin frame codec tests\n");
00037     RUN_TEST(test_frame_encode_basic);
00038     RUN_TEST(test_frame_decode_basic);
00039     RUN_TEST(test_frame_decode_invalid_header);
00040     uart_puts(uart0, "end frame codec tests\n");
00041
00042     // Frame build tests (pure software)
00043     uart_puts(uart0, "begin frame build tests\n");
00044     RUN_TEST(test_frame_build_success);
00045     RUN_TEST(test_frame_build_error);
00046     RUN_TEST(test_frame_build_info);
00047     uart_puts(uart0, "end frame build tests\n");
00048
00049     // Converter tests (pure software)
00050     uart_puts(uart0, "begin converter tests\n");
00051     RUN_TEST(test_operation_type_conversion);
00052     RUN_TEST(test_value_unit_type_conversion);
00053     RUN_TEST(test_exception_type_conversion);
00054     RUN_TEST(test_hex_string_conversion);
00055     RUN_TEST(test_error_code_conversion);
00056     uart_puts(uart0, "end converter tests\n");
00057
00058     // Command handler tests (pure software)
00059     uart_puts(uart0, "begin command handler tests\n");
00060     RUN_TEST(test_command_handler_get_operation);
00061     RUN_TEST(test_command_handler_set_operation);
00062     RUN_TEST(test_command_handler_invalid_operation);
00063     uart_puts(uart0, "end command handler tests\n");
00064
00065     return UNITY_END();
00066 }
```



# Index

\_BH1750\_DEFAULT\_MTREG  
  BH1750.h, 292

\_BH1750\_DEVICE\_ID  
  BH1750.h, 291

\_BH1750\_MTREG\_MAX  
  BH1750.h, 291

\_BH1750\_MTREG\_MIN  
  BH1750.h, 291

\_\_attribute\_\_  
  Event Manager, 48, 50

\_filterRes  
  INA3221, 146

\_i2c  
  INA3221, 146

\_i2c\_addr  
  BH1750, 78  
  INA3221, 146

\_masken\_reg  
  INA3221, 146

\_read  
  INA3221, 142

\_shuntRes  
  INA3221, 146

\_write  
  INA3221, 143

~EventManager  
  EventManager, 124

~ISensor  
  ISensor, 147

ACCEL\_X  
  ISensor.h, 312

ACCEL\_Y  
  ISensor.h, 312

ACCEL\_Z  
  ISensor.h, 312

ADDR\_SDO\_HIGH  
  BME280, 85

ADDR\_SDO\_LOW  
  BME280, 85

address  
  HMC5883L, 137

Alert Functions, 63

  enable\_alerts, 65

  get\_crit\_alert, 66

  get\_power\_valid\_alert, 68

  get\_warn\_alert, 66

  set\_alert\_latch, 68

  set\_crit\_alert\_limit, 64

  set\_power\_valid\_limit, 65

altitude  
  TelemetryRecord, 171

ANSI\_BLUE  
  utils.h, 341

ANSI\_CYAN  
  utils.h, 341

ANSI\_GREEN  
  utils.h, 341

ANSI\_RED  
  utils.h, 341

ANSI\_RESET  
  utils.h, 341

ANSI\_YELLOW  
  utils.h, 341

avg\_mode  
  INA3221::conf\_reg\_t, 98

base64\_chars  
  utils.cpp, 337

base64\_encode  
  utils.cpp, 336

  utils.h, 344

battery\_voltage  
  TelemetryRecord, 168

bcd\_to\_bin  
  DS3231, 115

begin  
  BH1750, 76

  Configuration Functions, 52

BH1750, 75

  \_i2c\_addr, 78

  begin, 76

  BH1750, 76

  configure, 76

  CONTINUOUS\_HIGH\_RES\_MODE, 76

  CONTINUOUS\_HIGH\_RES\_MODE\_2, 76

  CONTINUOUS\_LOW\_RES\_MODE, 76

  get\_light\_level, 77

  Mode, 75

  ONE\_TIME\_HIGH\_RES\_MODE, 76

  ONE\_TIME\_HIGH\_RES\_MODE\_2, 76

  ONE\_TIME\_LOW\_RES\_MODE, 76

  POWER\_ON, 76

  RESET, 76

  UNCONFIGURED\_POWER\_DOWN, 76

  write8, 77

BH1750.h  
  \_BH1750\_DEFAULT\_MTREG, 292

  \_BH1750\_DEVICE\_ID, 291

\_BH1750\_MTREG\_MAX, 291  
 \_BH1750\_MTREG\_MIN, 291  
 BH1750Wrapper, 78  
 BH1750Wrapper, 79  
 configure, 80  
 get\_address, 80  
 get\_i2c\_addr, 79  
 get\_type, 80  
 init, 79  
 initialized\_, 80  
 is\_initialized, 80  
 read\_data, 79  
 sensor\_, 80  
 bin\_to\_bcd  
 DS3231, 114  
 BME280, 81  
 ADDR\_SDO\_HIGH, 85  
 ADDR\_SDO\_LOW, 85  
 BME280, 83  
 calib\_params, 85  
 configure\_sensor, 84  
 convert\_humidity, 84  
 convert\_pressure, 84  
 convert\_temperature, 83  
 device\_addr, 85  
 get\_calibration\_parameters, 84  
 i2c\_port, 85  
 init, 83  
 initialized\_, 85  
 NUM\_CALIB\_PARAMS, 90  
 read\_raw\_all, 83  
 REG\_CONFIG, 85  
 REG\_CTRL\_HUM, 86  
 REG\_CTRL\_MEAS, 86  
 REG\_DIG\_H1, 89  
 REG\_DIG\_H2, 89  
 REG\_DIG\_H3, 90  
 REG\_DIG\_H4, 90  
 REG\_DIG\_H5, 90  
 REG\_DIG\_H6, 90  
 REG\_DIG\_P1\_LSB, 87  
 REG\_DIG\_P1\_MSB, 87  
 REG\_DIG\_P2\_LSB, 87  
 REG\_DIG\_P2\_MSB, 87  
 REG\_DIG\_P3\_LSB, 88  
 REG\_DIG\_P3\_MSB, 88  
 REG\_DIG\_P4\_LSB, 88  
 REG\_DIG\_P4\_MSB, 88  
 REG\_DIG\_P5\_LSB, 88  
 REG\_DIG\_P5\_MSB, 88  
 REG\_DIG\_P6\_LSB, 88  
 REG\_DIG\_P6\_MSB, 88  
 REG\_DIG\_P7\_LSB, 89  
 REG\_DIG\_P7\_MSB, 89  
 REG\_DIG\_P8\_LSB, 89  
 REG\_DIG\_P8\_MSB, 89  
 REG\_DIG\_P9\_LSB, 89  
 REG\_DIG\_P9\_MSB, 89

REG\_DIG\_T1\_LSB, 86  
 REG\_DIG\_T1\_MSB, 86  
 REG\_DIG\_T2\_LSB, 87  
 REG\_DIG\_T2\_MSB, 87  
 REG\_DIG\_T3\_LSB, 87  
 REG\_DIG\_T3\_MSB, 87  
 REG\_HUMIDITY\_MSB, 86  
 REG\_PRESSURE\_MSB, 86  
 REG\_RESET, 86  
 REG\_TEMPERATURE\_MSB, 86  
 reset, 83  
 t\_fine, 85  
 BME280CalibParam, 90  
 dig\_h1, 93  
 dig\_h2, 93  
 dig\_h3, 93  
 dig\_h4, 93  
 dig\_h5, 93  
 dig\_h6, 93  
 dig\_p1, 91  
 dig\_p2, 92  
 dig\_p3, 92  
 dig\_p4, 92  
 dig\_p5, 92  
 dig\_p6, 92  
 dig\_p7, 92  
 dig\_p8, 92  
 dig\_p9, 92  
 dig\_t1, 91  
 dig\_t2, 91  
 dig\_t3, 91  
 BME280Wrapper, 94  
 BME280Wrapper, 95  
 configure, 96  
 get\_address, 96  
 get\_type, 95  
 init, 95  
 initialized\_, 96  
 is\_initialized, 95  
 read\_data, 95  
 sensor\_, 96  
 BOOL  
 protocol.h, 228  
 BOOT  
 Event Manager, 47  
 buffer  
 main.cpp, 350  
 buffer\_index  
 main.cpp, 350  
 BUFFER\_SIZE  
 pin\_config.h, 269  
 BUILD\_NUMBER  
 build\_number.h, 173  
 build\_number.h, 173  
 BUILD\_NUMBER, 173  
 bus\_conv\_time  
 INA3221::conf\_reg\_t, 97  
 calculate\_checksum

utils.cpp, 336  
utils.h, 344  
calib\_params  
    BME280, 85  
century  
    ds3231\_data\_t, 119  
ch1\_en  
    INA3221::conf\_reg\_t, 98  
ch2\_en  
    INA3221::conf\_reg\_t, 98  
ch3\_en  
    INA3221::conf\_reg\_t, 98  
CHANGED  
    Event Manager, 48  
charge\_current\_solar  
    TelemetryRecord, 168  
charge\_current\_usb  
    TelemetryRecord, 168  
CHARGE\_SOLAR  
    power\_commands.cpp, 203  
CHARGE\_TOTAL  
    power\_commands.cpp, 203  
CHARGE\_USB  
    power\_commands.cpp, 202  
charging\_solar\_active\_  
    PowerManager, 161  
charging\_usb\_active\_  
    PowerManager, 161  
check\_power\_alerts  
    PowerManager, 160  
CLOCK  
    Event Manager, 46  
Clock Commands, 1  
Clock Management Commands, 11  
    handle\_clock\_sync\_interval, 13  
    handle\_get\_last\_sync\_time, 14  
    handle\_time, 11  
    handle\_timezone\_offset, 12  
    systemClock, 15  
clock\_commands.cpp  
    CLOCK\_GROUP, 186  
    CLOCK\_SYNC\_INTERVAL, 186  
    LAST\_SYNC\_TIME, 186  
    TIME, 186  
    TIMEZONE\_OFFSET, 186  
clock\_enable  
    DS3231, 106  
CLOCK\_GROUP  
    clock\_commands.cpp, 186  
clock\_mutex\_  
    DS3231, 116  
CLOCK\_SYNC\_INTERVAL  
    clock\_commands.cpp, 186  
ClockEvent  
    Event Manager, 48  
collect\_gps\_data  
    gps\_collector.cpp, 257  
    gps\_collector.h, 260  
collect\_telemetry  
    Telemetry Manager, 70  
command  
    Frame, 133  
Command System, 15  
    command\_handlers, 17  
    CommandHandler, 15  
    CommandMap, 15  
    execute\_command, 16  
command\_handlers  
    Command System, 17  
CommandAccessLevel  
    protocol.h, 227  
CommandHandler  
    Command System, 15  
    frame.cpp, 223  
CommandMap  
    Command System, 15  
COMMS  
    Event Manager, 46  
CommsEvent  
    Event Manager, 47  
communication.cpp  
    initialize\_radio, 214  
    interval, 215  
    lastPrintTime, 215  
    lastReceiveTime, 215  
    lastSendTime, 215  
    msgCount, 215  
    outgoing, 215  
communication.h  
    determine\_unit, 221  
    handle\_uart\_input, 218  
    initialize\_radio, 217  
    on\_receive, 218  
    send\_frame\_lora, 220  
    send\_frame\_uart, 219  
    send\_message, 219  
    split\_and\_send\_message, 221  
Configuration Functions, 51  
    begin, 52  
    get\_die\_id, 54  
    get\_manufacturer\_id, 53  
    INA3221, 52  
    read\_register, 55  
    reset, 53  
    set\_averaging\_mode, 59  
    set\_bus\_conversion\_time, 59  
    set\_bus\_measurement\_disable, 58  
    set\_bus\_measurement\_enable, 58  
    set\_mode\_continuous, 56  
    set\_mode\_power\_down, 55  
    set\_mode\_triggered, 56  
    set\_shunt\_conversion\_time, 60  
    set\_shunt\_measurement\_disable, 57  
    set\_shunt\_measurement\_enable, 57  
configure  
    BH1750, 76

BH1750Wrapper, 80  
 BME280Wrapper, 96  
 HMC5883LWrapper, 139  
 ISensor, 147  
 MPU6050Wrapper, 152  
 PowerManager, 159  
 configure\_sensor  
     BME280, 84  
 CONTINUOUS\_HIGH\_RES\_MODE  
     BH1750, 76  
 CONTINUOUS\_HIGH\_RES\_MODE\_2  
     BH1750, 76  
 CONTINUOUS\_LOW\_RES\_MODE  
     BH1750, 76  
 conv\_ready  
     INA3221::masken\_reg\_t, 149  
 convert\_humidity  
     BME280, 84  
 convert\_pressure  
     BME280, 84  
 convert\_temperature  
     BME280, 83  
 core1\_entry  
     main.cpp, 347  
 CORE1\_START  
     Event Manager, 47  
 CORE1\_STOP  
     Event Manager, 47  
 course  
     TelemetryRecord, 170  
 create\_test\_frame  
     test\_frame\_common.h, 369  
 crit\_alert\_ch1  
     INA3221::masken\_reg\_t, 150  
 crit\_alert\_ch2  
     INA3221::masken\_reg\_t, 150  
 crit\_alert\_ch3  
     INA3221::masken\_reg\_t, 149  
 crit\_alert\_latch\_en  
     INA3221::masken\_reg\_t, 150  
 DATA\_READY  
     Event Manager, 48  
 date  
     ds3231\_data\_t, 119  
     TelemetryRecord, 171  
 DATETIME  
     protocol.h, 228  
 day  
     ds3231\_data\_t, 118  
 days\_of\_week  
     DS3231.h, 183  
 DEBUG  
     utils.h, 342  
 DEBUG\_UART\_BAUD\_RATE  
     pin\_config.h, 267  
 DEBUG\_UART\_PORT  
     pin\_config.h, 267  
 DEBUG\_UART\_RX\_PIN  
     pin\_config.h, 268  
 DEBUG\_UART\_TX\_PIN  
     pin\_config.h, 268  
 DEFAULT\_FLUSH\_THRESHOLD  
     telemetry\_manager.cpp, 324  
 DEFAULT\_SAMPLE\_INTERVAL\_MS  
     telemetry\_manager.cpp, 324  
 DELIMITER  
     protocol.h, 234  
 determine\_unit  
     communication.h, 221  
 device\_addr  
     BME280, 85  
 Diagnostic Commands, 17  
     g\_pending\_bootloader\_reset, 21  
     handle\_enter\_bootloader\_mode, 20  
     handle\_get\_build\_version, 18  
     handle\_get\_commands\_list, 18  
     handle\_verbosity, 19  
 dig\_h1  
     BME280CalibParam, 93  
 dig\_h2  
     BME280CalibParam, 93  
 dig\_h3  
     BME280CalibParam, 93  
 dig\_h4  
     BME280CalibParam, 93  
 dig\_h5  
     BME280CalibParam, 93  
 dig\_h6  
     BME280CalibParam, 93  
 dig\_p1  
     BME280CalibParam, 91  
 dig\_p2  
     BME280CalibParam, 92  
 dig\_p3  
     BME280CalibParam, 92  
 dig\_p4  
     BME280CalibParam, 92  
 dig\_p5  
     BME280CalibParam, 92  
 dig\_p6  
     BME280CalibParam, 92  
 dig\_p7  
     BME280CalibParam, 92  
 dig\_p8  
     BME280CalibParam, 92  
 dig\_p9  
     BME280CalibParam, 92  
 dig\_t1  
     BME280CalibParam, 91  
 dig\_t2  
     BME280CalibParam, 91  
 dig\_t3  
     BME280CalibParam, 91  
 direction  
     Frame, 133  
 discharge\_current

TelemetryRecord, 169  
DRAW\_TOTAL  
power\_commands.cpp, 203  
DS3231, 98  
  bcd\_to\_bin, 115  
  bin\_to\_bcd, 114  
  clock\_enable, 106  
  clock\_mutex\_, 116  
  DS3231, 100  
  ds3231\_addr, 116  
  get\_clock\_sync\_interval, 108  
  get\_last\_sync\_time, 109  
  get\_local\_time, 110  
  get\_time, 101  
  get\_timezone\_offset, 107  
  get\_unix\_time, 105  
  i2c, 116  
  i2c\_read\_reg, 112  
  i2c\_write\_reg, 113  
  is\_sync\_needed, 111  
  last\_sync\_time\_, 117  
  read\_temperature, 102  
  set\_clock\_sync\_interval, 108  
  set\_time, 100  
  set\_timezone\_offset, 107  
  set\_unix\_time, 104  
  sync\_clock\_with\_gps, 111  
  sync\_interval\_minutes\_, 117  
  timezone\_offset\_minutes\_, 117  
  update\_last\_sync\_time, 109  
DS3231.h  
  days\_of\_week, 183  
  DS3231\_CONTROL\_REG, 182  
  DS3231\_CONTROL\_STATUS\_REG, 183  
  DS3231\_DATE\_REG, 182  
  DS3231\_DAY\_REG, 182  
  DS3231\_DEVICE\_ADDRESS, 181  
  DS3231\_HOURS\_REG, 182  
  DS3231\_MINUTES\_REG, 181  
  DS3231\_MONTH\_REG, 182  
  DS3231\_SECONDS\_REG, 181  
  DS3231\_TEMPERATURE\_LSB\_REG, 183  
  DS3231\_TEMPERATURE\_MSB\_REG, 183  
  DS3231\_YEAR\_REG, 182  
  FRIDAY, 183  
  MONDAY, 183  
  SATURDAY, 183  
  SUNDAY, 183  
  THURSDAY, 183  
  TUESDAY, 183  
  WEDNESDAY, 183  
ds3231\_addr  
  DS3231, 116  
DS3231\_CONTROL\_REG  
  DS3231.h, 182  
DS3231\_CONTROL\_STATUS\_REG  
  DS3231.h, 183  
ds3231\_data\_t, 117  
  century, 119  
  date, 119  
  day, 118  
  hours, 118  
  minutes, 118  
  month, 119  
  seconds, 118  
  year, 119  
  DS3231\_DATE\_REG  
    DS3231.h, 182  
  DS3231\_DAY\_REG  
    DS3231.h, 182  
  DS3231\_DEVICE\_ADDRESS  
    DS3231.h, 181  
  DS3231\_HOURS\_REG  
    DS3231.h, 182  
  DS3231\_MINUTES\_REG  
    DS3231.h, 181  
  DS3231\_MONTH\_REG  
    DS3231.h, 182  
  DS3231\_SECONDS\_REG  
    DS3231.h, 181  
  DS3231\_TEMPERATURE\_LSB\_REG  
    DS3231.h, 183  
  DS3231\_TEMPERATURE\_MSB\_REG  
    DS3231.h, 183  
  DS3231\_YEAR\_REG  
    DS3231.h, 182  
emit  
  EventEmitter, 120  
enable\_alerts  
  Alert Functions, 65  
ENVIRONMENT  
  ISensor.h, 312  
ERR  
  protocol.h, 227  
ERROR  
  Event Manager, 48  
  utils.h, 342  
error\_code\_to\_string  
  protocol.h, 229  
  utils\_converters.cpp, 245  
ErrorCode  
  protocol.h, 226  
EVENT  
  utils.h, 342  
event  
  event\_manager.h, 254  
  EventLog, 123  
Event Commands, 21  
  handle\_get\_event\_count, 23  
  handle\_get\_last\_events, 22  
Event Manager, 44  
  \_\_attribute\_\_, 48, 50  
  BOOT, 47  
  CHANGED, 48  
  CLOCK, 46  
  ClockEvent, 48

COMMS, 46  
 CommsEvent, 47  
 CORE1\_START, 47  
 CORE1\_STOP, 47  
 DATA\_READY, 48  
 ERROR, 48  
 EventGroup, 46  
 eventLogId, 50  
 eventManager, 50  
 get\_event, 49  
 GPS, 46  
 GPS\_SYNC, 48  
 GPS\_SYNC\_DATA\_NOT\_READY, 48  
 GPSEvent, 47  
 LOCK, 48  
 log\_event, 48  
 LOST, 48  
 LOW\_BATTERY, 47  
 MSG RECEIVED, 47  
 MSG SENT, 47  
 OVERCHARGE, 47  
 PASS THROUGH END, 48  
 PASS THROUGH START, 48  
 POWER, 46  
 POWER\_FALLING, 47  
 POWER\_NORMAL, 47  
 POWER\_OFF, 48  
 POWER\_ON, 48  
 PowerEvent, 47  
 RADIO\_ERROR, 47  
 RADIO\_INIT, 47  
 SHUTDOWN, 47  
 SOLAR\_ACTIVE, 47  
 SOLAR\_INACTIVE, 47  
 SYSTEM, 46  
 systemClock, 50  
 SystemEvent, 46  
 UART\_ERROR, 47  
 USB\_CONNECTED, 47  
 USB\_DISCONNECTED, 47  
 WATCHDOG\_RESET, 47  
 EVENT\_BUFFER\_SIZE  
     event\_manager.h, 253  
 EVENT\_FLUSH\_THRESHOLD  
     event\_manager.h, 253  
 EVENT\_LOG\_FILE  
     event\_manager.h, 253  
 event\_manager.h  
     event, 254  
     EVENT\_BUFFER\_SIZE, 253  
     EVENT\_FLUSH\_THRESHOLD, 253  
     EVENT\_LOG\_FILE, 253  
     group, 254  
     id, 253  
     timestamp, 253  
     to\_string, 253  
 eventCount  
     EventManager, 126  
 EventEmitter, 119  
     emit, 120  
 EventGroup  
     Event Manager, 46  
 EventLog, 121  
     event, 123  
     group, 122  
     id, 122  
     timestamp, 122  
     to\_string, 122  
 eventLogId  
     Event Manager, 50  
 EventManager, 123  
     ~EventManager, 124  
     eventCount, 126  
     EventManager, 124  
     eventMutex, 127  
     events, 126  
     eventsSinceFlush, 127  
     get\_event\_count, 125  
     init, 125  
     load\_from\_storage, 126  
     nextEventId, 127  
     save\_to\_storage, 125  
     writeIndex, 127  
 eventManager  
     Event Manager, 50  
 EventManagerImpl, 128  
     EventManagerImpl, 130  
     load\_from\_storage, 130  
     save\_to\_storage, 130  
 eventMutex  
     EventManager, 127  
 eventRegister  
     frame.cpp, 223  
 events  
     EventManager, 126  
 eventsSinceFlush  
     EventManager, 127  
 exception\_type\_to\_string  
     protocol.h, 228  
     utils\_converters.cpp, 242  
 ExceptionType  
     protocol.h, 228  
 execute\_command  
     Command System, 16  
 FAIL\_TO\_SET  
     protocol.h, 227  
 FALL\_RATE\_THRESHOLD  
     PowerManager, 161  
 FALLING\_TREND\_REQUIRED  
     PowerManager, 161  
 fd  
     FileHandle, 131  
 FileHandle, 131  
     fd, 131  
     is\_open, 131  
 fix\_quality

TelemetryRecord, 171  
flush\_telemetry  
    Telemetry Manager, 71  
flush\_threshold  
    telemetry\_manager.cpp, 325  
footer  
    Frame, 134  
Frame, 132  
    command, 133  
    direction, 133  
    footer, 134  
    group, 133  
    header, 133  
    operationType, 133  
    unit, 134  
    value, 134  
Frame Handling, 39  
    frame\_build, 43  
    frame\_decode, 41  
    frame\_encode, 40  
    frame\_process, 42  
frame.cpp  
    CommandHandler, 223  
    eventRegister, 223  
FRAME\_BEGIN  
    protocol.h, 234  
frame\_build  
    Frame Handling, 43  
frame\_decode  
    Frame Handling, 41  
frame\_encode  
    Frame Handling, 40  
FRAME\_END  
    protocol.h, 234  
frame\_process  
    Frame Handling, 42  
FRIDAY  
    DS3231.h, 183  
fs\_close\_file  
    storage.h, 321  
fs\_file\_exists  
    storage.h, 321  
fs\_init  
    storage.cpp, 317  
    storage.h, 319  
fs\_open\_file  
    storage.h, 320  
fs\_read\_file  
    storage.h, 320  
fs\_write\_file  
    storage.h, 320  
g\_pending\_bootloader\_reset  
    Diagnostic Commands, 21  
g\_uart\_verbosity  
    utils.cpp, 337  
    utils.h, 345  
GET  
    protocol.h, 227  
get\_address  
    BH1750Wrapper, 80  
    BME280Wrapper, 96  
    HMC5883LWrapper, 139  
    ISensor, 148  
get\_available\_sensors  
    SensorWrapper, 166  
get\_calibration\_parameters  
    BME280, 84  
get\_clock\_sync\_interval  
    DS3231, 108  
get\_crit\_alert  
    Alert Functions, 66  
get\_current  
    INA3221, 145  
get\_current\_charge\_solar  
    PowerManager, 158  
get\_current\_charge\_total  
    PowerManager, 158  
get\_current\_charge\_usb  
    PowerManager, 158  
get\_current\_draw  
    PowerManager, 159  
get\_current\_ma  
    Measurement Functions, 62  
get\_die\_id  
    Configuration Functions, 54  
get\_event  
    Event Manager, 49  
get\_event\_count  
    EventManager, 125  
get\_gga\_tokens  
    NMEAData, 154  
get\_i2c\_addr  
    BH1750Wrapper, 79  
get\_instance  
    SensorWrapper, 163  
get\_last\_sync\_time  
    DS3231, 109  
get\_level\_color  
    utils.cpp, 332  
get\_level\_prefix  
    utils.cpp, 333  
get\_light\_level  
    BH1750, 77  
get\_local\_time  
    DS3231, 110  
get\_manufacturer\_id  
    Configuration Functions, 53  
get\_power\_valid\_alert  
    Alert Functions, 68  
get\_rmc\_tokens  
    NMEAData, 154  
get\_sensor  
    SensorWrapper, 165  
get\_shunt\_voltage  
    Measurement Functions, 61  
get\_telemetry\_flush\_threshold

Telemetry Manager, 74  
 get\_telemetry\_sample\_interval  
     Telemetry Manager, 73  
 get\_time  
     DS3231, 101  
 get\_timezone\_offset  
     DS3231, 107  
 get\_type  
     BH1750Wrapper, 80  
     BME280Wrapper, 95  
     HMC5883LWrapper, 139  
     ISensor, 147  
     MPU6050Wrapper, 152  
 get\_unix\_time  
     DS3231, 105  
     NMEAData, 155  
 get\_voltage  
     Measurement Functions, 62  
 get\_voltage\_5v  
     PowerManager, 159  
 get\_voltage\_battery  
     PowerManager, 159  
 get\_warn\_alert  
     Alert Functions, 66  
 GGA\_DATA\_COMMAND  
     gps\_commands.cpp, 198  
 gga\_mutex\_  
     NMEAData, 156  
 gga\_tokens\_  
     NMEAData, 155  
 GPS  
     Event Manager, 46  
 GPS Commands, 24  
     handle\_enable\_gps\_uart\_passthrough, 25  
     handle\_get\_gga\_data, 27  
     handle\_get\_rmc\_data, 26  
     handle\_gps\_power\_status, 24  
 gps\_collector.cpp  
     collect\_gps\_data, 257  
     MAX\_RAW\_DATA\_LENGTH, 257  
     nmea\_data, 258  
     pause\_gps\_collection, 258  
     splitString, 257  
 gps\_collector.h  
     collect\_gps\_data, 260  
 gps\_commands.cpp  
     GGA\_DATA\_COMMAND, 198  
     GPS\_GROUP, 198  
     PASSTHROUGH\_COMMAND, 198  
     POWER\_STATUS\_COMMAND, 198  
     RMC\_DATA\_COMMAND, 198  
 GPS\_GROUP  
     gps\_commands.cpp, 198  
 GPS\_POWER\_ENABLE\_PIN  
     pin\_config.h, 269  
 GPS\_SYNC  
     Event Manager, 48  
 GPS\_SYNC\_DATA\_NOT\_READY

    Event Manager, 48  
 GPS\_UART\_BAUD\_RATE  
     pin\_config.h, 268  
 GPS\_UART\_PORT  
     pin\_config.h, 268  
 GPS\_UART\_RX\_PIN  
     pin\_config.h, 269  
 GPS\_UART\_TX\_PIN  
     pin\_config.h, 268  
 GPSEvent  
     Event Manager, 47  
 group  
     event\_manager.h, 254  
     EventLog, 122  
     Frame, 133  
 GYRO\_X  
     ISensor.h, 312  
 GYRO\_Y  
     ISensor.h, 312  
 GYRO\_Z  
     ISensor.h, 312  
 handle\_clock\_sync\_interval  
     Clock Management Commands, 13  
 handle\_enable\_gps\_uart\_passthrough  
     GPS Commands, 25  
 handle\_enter\_bootloader\_mode  
     Diagnostic Commands, 20  
 handle\_get\_build\_version  
     Diagnostic Commands, 18  
 handle\_get\_commands\_list  
     Diagnostic Commands, 18  
 handle\_get\_current\_charge\_solar  
     Power Commands, 31  
 handle\_get\_current\_charge\_total  
     Power Commands, 32  
 handle\_get\_current\_charge\_usb  
     Power Commands, 31  
 handle\_get\_current\_draw  
     Power Commands, 33  
 handle\_get\_event\_count  
     Event Commands, 23  
 handle\_get\_gga\_data  
     GPS Commands, 27  
 handle\_get\_last\_events  
     Event Commands, 22  
 handle\_get\_last\_sync\_time  
     Clock Management Commands, 14  
 handle\_get\_power\_manager\_ids  
     Power Commands, 28  
 handle\_get\_rmc\_data  
     GPS Commands, 26  
 handle\_get\_sensor\_data  
     Sensor Commands, 34  
 handle\_get\_sensor\_list  
     Sensor Commands, 36  
 handle\_get\_voltage\_5v  
     Power Commands, 30  
 handle\_get\_voltage\_battery

Power Commands, 29  
handle\_gps\_power\_status  
    GPS Commands, 24  
handle\_list\_files  
    Storage Commands, 38  
handle\_mount  
    Storage Commands, 38  
handle\_sensor\_config  
    Sensor Commands, 35  
handle\_time  
    Clock Management Commands, 11  
handle\_timezone\_offset  
    Clock Management Commands, 12  
handle\_uart\_input  
    communication.h, 218  
    receive.cpp, 237  
handle\_verbosity  
    Diagnostic Commands, 19  
hardwareMocks.cpp  
    mock\_spi\_enabled, 373  
    mock\_spi\_read\_blocking, 372  
    mock\_spi\_write\_blocking, 372  
    mock\_uart\_enabled, 373  
    mock\_uart\_init, 372  
    mock\_uart\_puts, 372  
    spi\_output\_buffer, 373  
    uart\_output\_buffer, 373  
hardwareMocks.h  
    mock\_spi\_enabled, 375  
    mock\_spi\_read\_blocking, 375  
    mock\_spi\_write\_blocking, 375  
    mock\_uart\_enabled, 375  
    mock\_uart\_init, 375  
    mock\_uart\_puts, 375  
    spi\_output\_buffer, 376  
    uart\_output\_buffer, 375  
has\_valid\_time  
    NMEAData, 154  
header  
    Frame, 133  
hex\_string\_to\_bytes  
    protocol.h, 231  
    utils\_converters.cpp, 247  
HMC5883L, 134  
    address, 137  
    HMC5883L, 135  
    i2c, 137  
    init, 135  
    read, 135  
    read\_register, 136  
    write\_register, 136  
HMC5883LWrapper, 137  
    configure, 139  
    get\_address, 139  
    get\_type, 139  
    HMC5883LWrapper, 138  
    init, 139  
    initialized\_, 140  
is\_initialized, 139  
read\_data, 139  
sensor\_, 140  
hours  
    ds3231\_data\_t, 118  
HUMIDITY  
    ISensor.h, 312  
humidity  
    TelemetryRecord, 169  
i2c  
    DS3231, 116  
    HMC5883L, 137  
i2c\_port  
    BME280, 85  
i2c\_read\_reg  
    DS3231, 112  
i2c\_write\_reg  
    DS3231, 113  
id  
    event\_manager.h, 253  
    EventLog, 122  
IMU  
    ISensor.h, 312  
INA3221, 140  
    \_filterRes, 146  
    \_i2c, 146  
    \_i2c\_addr, 146  
    \_masken\_reg, 146  
    \_read, 142  
    \_shuntRes, 146  
    \_write, 143  
    Configuration Functions, 52  
    get\_current, 145  
INA3221 Power Monitor, 51  
INA3221.h  
    INA3221\_ADDR40\_GND, 280  
    INA3221\_ADDR41\_VCC, 280  
    INA3221\_ADDR42\_SDA, 280  
    INA3221\_ADDR43\_SCL, 280  
    ina3221\_addr\_t, 280  
    ina3221\_avg\_mode\_t, 282  
    INA3221\_CH1, 281  
    INA3221\_CH2, 281  
    INA3221\_CH3, 281  
    INA3221\_CH\_NUM, 282  
    ina3221\_ch\_t, 280  
    ina3221\_conv\_time\_t, 281  
    INA3221\_REG\_CH1\_BUSV, 281  
    INA3221\_REG\_CH1\_CRIT\_ALERT\_LIM, 281  
    INA3221\_REG\_CH1\_SHUNTV, 281  
    INA3221\_REG\_CH1\_WARNING\_ALERT\_LIM,  
        281  
    INA3221\_REG\_CH2\_BUSV, 281  
    INA3221\_REG\_CH2\_CRIT\_ALERT\_LIM, 281  
    INA3221\_REG\_CH2\_SHUNTV, 281  
    INA3221\_REG\_CH2\_WARNING\_ALERT\_LIM,  
        281  
    INA3221\_REG\_CH3\_BUSV, 281

INA3221\_REG\_CH3\_CRIT\_ALERT\_LIM, 281  
 INA3221\_REG\_CH3\_SHUNTV, 281  
 INA3221\_REG\_CH3\_WARNING\_ALERT\_LIM, 281  
 INA3221\_REG\_CONF, 281  
 INA3221\_REG\_CONF\_AVG\_1, 282  
 INA3221\_REG\_CONF\_AVG\_1024, 282  
 INA3221\_REG\_CONF\_AVG\_128, 282  
 INA3221\_REG\_CONF\_AVG\_16, 282  
 INA3221\_REG\_CONF\_AVG\_256, 282  
 INA3221\_REG\_CONF\_AVG\_4, 282  
 INA3221\_REG\_CONF\_AVG\_512, 282  
 INA3221\_REG\_CONF\_AVG\_64, 282  
 INA3221\_REG\_CONF\_CT\_1100US, 282  
 INA3221\_REG\_CONF\_CT\_140US, 282  
 INA3221\_REG\_CONF\_CT\_204US, 282  
 INA3221\_REG\_CONF\_CT\_2116US, 282  
 INA3221\_REG\_CONF\_CT\_332US, 282  
 INA3221\_REG\_CONF\_CT\_4156US, 282  
 INA3221\_REG\_CONF\_CT\_588US, 282  
 INA3221\_REG\_CONF\_CT\_8244US, 282  
 INA3221\_REG\_DIE\_ID, 281  
 INA3221\_REG\_MANUF\_ID, 281  
 INA3221\_REG\_MASK\_ENABLE, 281  
 INA3221\_REG\_PWR\_VALID\_HI\_LIM, 281  
 INA3221\_REG\_PWR\_VALID\_LO\_LIM, 281  
 INA3221\_REG\_SHUNTV\_SUM, 281  
 INA3221\_REG\_SHUNTV\_SUM\_LIM, 281  
 ina3221\_reg\_t, 281  
 SHUNT\_VOLTAGE\_LSB\_UV, 282  
 INA3221::conf\_reg\_t, 96  
 avg\_mode, 98  
 bus\_conv\_time, 97  
 ch1\_en, 98  
 ch2\_en, 98  
 ch3\_en, 98  
 mode\_bus\_en, 97  
 mode\_continious\_en, 97  
 mode\_shunt\_en, 97  
 reset, 98  
 shunt\_conv\_time, 97  
 INA3221::masken\_reg\_t, 148  
 conv\_ready, 149  
 crit\_alert\_ch1, 150  
 crit\_alert\_ch2, 150  
 crit\_alert\_ch3, 149  
 crit\_alert\_latch\_en, 150  
 pwr\_valid\_alert, 149  
 reserved, 150  
 shunt\_sum\_alert, 149  
 shunt\_sum\_en\_ch1, 150  
 shunt\_sum\_en\_ch2, 150  
 shunt\_sum\_en\_ch3, 150  
 timing\_ctrl\_alert, 149  
 warn\_alert\_ch1, 149  
 warn\_alert\_ch2, 149  
 warn\_alert\_ch3, 149  
 warn\_alert\_latch\_en, 150  
 ina3221\_  
     PowerManager, 161  
 INA3221\_ADDR40\_GND  
     INA3221.h, 280  
 INA3221\_ADDR41\_VCC  
     INA3221.h, 280  
 INA3221\_ADDR42\_SDA  
     INA3221.h, 280  
 INA3221\_ADDR43\_SCL  
     INA3221.h, 280  
 ina3221\_addr\_t  
     INA3221.h, 280  
 ina3221\_avg\_mode\_t  
     INA3221.h, 282  
 INA3221\_CH1  
     INA3221.h, 281  
 INA3221\_CH2  
     INA3221.h, 281  
 INA3221\_CH3  
     INA3221.h, 281  
 INA3221\_CH\_NUM  
     INA3221.h, 282  
 ina3221\_ch\_t  
     INA3221.h, 280  
 ina3221\_conv\_time\_t  
     INA3221.h, 281  
 INA3221\_REG\_CH1\_BUSV  
     INA3221.h, 281  
 INA3221\_REG\_CH1\_CRIT\_ALERT\_LIM  
     INA3221.h, 281  
 INA3221\_REG\_CH1\_SHUNTV  
     INA3221.h, 281  
 INA3221\_REG\_CH1\_WARNING\_ALERT\_LIM  
     INA3221.h, 281  
 INA3221\_REG\_CH2\_BUSV  
     INA3221.h, 281  
 INA3221\_REG\_CH2\_CRIT\_ALERT\_LIM  
     INA3221.h, 281  
 INA3221\_REG\_CH2\_SHUNTV  
     INA3221.h, 281  
 INA3221\_REG\_CH2\_WARNING\_ALERT\_LIM  
     INA3221.h, 281  
 INA3221\_REG\_CH3\_BUSV  
     INA3221.h, 281  
 INA3221\_REG\_CH3\_CRIT\_ALERT\_LIM  
     INA3221.h, 281  
 INA3221\_REG\_CH3\_SHUNTV  
     INA3221.h, 281  
 INA3221\_REG\_CH3\_WARNING\_ALERT\_LIM  
     INA3221.h, 281  
 INA3221\_REG\_CONF  
     INA3221.h, 281  
 INA3221\_REG\_CONF\_AVG\_1  
     INA3221.h, 282  
 INA3221\_REG\_CONF\_AVG\_1024  
     INA3221.h, 282  
 INA3221\_REG\_CONF\_AVG\_128  
     INA3221.h, 282

INA3221\_REG\_CONF\_AVG\_16  
  INA3221.h, 282  
INA3221\_REG\_CONF\_AVG\_256  
  INA3221.h, 282  
INA3221\_REG\_CONF\_AVG\_4  
  INA3221.h, 282  
INA3221\_REG\_CONF\_AVG\_512  
  INA3221.h, 282  
INA3221\_REG\_CONF\_AVG\_64  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_1100US  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_140US  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_204US  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_2116US  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_332US  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_4156US  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_588US  
  INA3221.h, 282  
INA3221\_REG\_CONF\_CT\_8244US  
  INA3221.h, 282  
INA3221\_REG\_DIE\_ID  
  INA3221.h, 281  
INA3221\_REG\_MANUF\_ID  
  INA3221.h, 281  
INA3221\_REG\_MASK\_ENABLE  
  INA3221.h, 281  
INA3221\_REG\_PWR\_VALID\_HI\_LIM  
  INA3221.h, 281  
INA3221\_REG\_PWR\_VALID\_LO\_LIM  
  INA3221.h, 281  
INA3221\_REG\_SHUNTV\_SUM  
  INA3221.h, 281  
INA3221\_REG\_SHUNTV\_SUM\_LIM  
  INA3221.h, 281  
ina3221\_reg\_t  
  INA3221.h, 281  
includes.h, 174  
INFO  
  utils.h, 342  
init  
  BH1750Wrapper, 79  
  BME280, 83  
  BME280Wrapper, 95  
  EventManager, 125  
  HMC5883L, 135  
  HMC5883LWrapper, 139  
  ISensor, 147  
  MPU6050Wrapper, 152  
init\_systems  
  main.cpp, 348  
initialize  
  PowerManager, 158  
  initialize\_radio  
    communication.cpp, 214  
    communication.h, 217  
  initialized\_  
    BH1750Wrapper, 80  
    BME280, 85  
    BME280Wrapper, 96  
    HMC5883LWrapper, 140  
    MPU6050Wrapper, 153  
    PowerManager, 161  
  Interface  
    protocol.h, 228  
  INTERNAL\_FAIL\_TO\_READ  
    protocol.h, 227  
interval  
  communication.cpp, 215  
INVALID\_FORMAT  
  protocol.h, 227  
INVALID\_OPERATION  
  protocol.h, 227, 228  
INVALID\_PARAM  
  protocol.h, 228  
INVALID\_VALUE  
  protocol.h, 227  
is\_charging\_solar  
  PowerManager, 159  
is\_charging\_usb  
  PowerManager, 159  
is\_initialized  
  BH1750Wrapper, 80  
  BME280Wrapper, 95  
  HMC5883LWrapper, 139  
  ISensor, 147  
  MPU6050Wrapper, 152  
is\_open  
  FileHandle, 131  
is\_sync\_needed  
  DS3231, 111  
is\_telemetry\_collection\_time  
  Telemetry Manager, 72  
is\_telemetry\_flush\_time  
  Telemetry Manager, 73  
ISensor, 146  
  ~ISensor, 147  
  configure, 147  
  get\_address, 148  
  get\_type, 147  
  init, 147  
  is\_initialized, 147  
  read\_data, 147  
ISensor.h  
  ACCEL\_X, 312  
  ACCEL\_Y, 312  
  ACCEL\_Z, 312  
  ENVIRONMENT, 312  
  GYRO\_X, 312  
  GYRO\_Y, 312  
  GYRO\_Z, 312

HUMIDITY, 312  
IMU, 312  
LIGHT, 312  
LIGHT\_LEVEL, 312  
MAG\_FIELD\_X, 312  
MAG\_FIELD\_Y, 312  
MAG\_FIELD\_Z, 312  
MAGNETOMETER, 312  
PRESSURE, 312  
SensorDataTypelIdentifier, 312  
SensorType, 312  
TEMPERATURE, 312

last\_frame\_sent  
test\_command\_handlers.cpp, 357

LAST\_SYNC\_TIME  
clock\_commands.cpp, 186

last\_sync\_time\_  
DS3231, 117

lastPrintTime  
communication.cpp, 215

lastReceiveTime  
communication.cpp, 215

lastSendTime  
communication.cpp, 215

lat\_dir  
TelemetryRecord, 170

latitude  
TelemetryRecord, 170

lib/clock/DS3231.cpp, 175

lib/clock/DS3231.h, 180, 184

lib/comms/commands/clock\_commands.cpp, 185, 187

lib/comms/commands/commands.cpp, 189, 190

lib/comms/commands/commands.h, 191, 193

lib/comms/commands/diagnostic\_commands.cpp, 193, 194

lib/comms/commands/event\_commands.cpp, 196

lib/comms/commands/gps\_commands.cpp, 197, 199

lib/comms/commands/power\_commands.cpp, 201, 203

lib/comms/commands/sensor\_commands.cpp, 205, 207

lib/comms/commands/storage\_commands.cpp, 210, 212

lib/comms/communication.cpp, 213, 216

lib/comms/communication.h, 216, 221

lib/comms/frame.cpp, 222, 223

lib/comms/protocol.h, 225, 234

lib/comms/receive.cpp, 235, 238

lib/comms/send.cpp, 239, 241

lib/comms/utils\_converters.cpp, 241, 248

lib/eventman/event\_manager.cpp, 249, 250

lib/eventman/event\_manager.h, 251, 254

lib/location/gps\_collector.cpp, 256, 259

lib/location/gps\_collector.h, 259, 261

lib/location/NMEA/NMEA\_data.cpp, 261, 262

lib/location/NMEA/NMEA\_data.h, 263, 264

lib/pin\_config.cpp, 265, 266

lib/pin\_config.h, 266, 273

lib/powerman/INA3221/INA3221.cpp, 273, 274

lib/powerman/INA3221/INA3221.h, 278, 283

lib/powerman/PowerManager.cpp, 285  
lib/powerman/PowerManager.h, 287, 288

lib/sensors/BH1750/BH1750.cpp, 289  
lib/sensors/BH1750/BH1750.h, 290, 292

lib/sensors/BH1750/BH1750\_WRAPPER.cpp, 292, 293  
lib/sensors/BH1750/BH1750\_WRAPPER.h, 294, 295

lib/sensors/BME280/BME280.cpp, 295, 296  
lib/sensors/BME280/BME280.h, 299, 300

lib/sensors/BME280/BME280\_WRAPPER.cpp, 302  
lib/sensors/BME280/BME280\_WRAPPER.h, 303

lib/sensors/HMC5883L/HMC5883L.cpp, 304  
lib/sensors/HMC5883L/HMC5883L.h, 305, 306

lib/sensors/HMC5883L/HMC5883L\_WRAPPER.cpp, 306, 307

lib/sensors/HMC5883L/HMC5883L\_WRAPPER.h, 308, 309

lib/sensors/ISensor.cpp, 309, 310  
lib/sensors/ISensor.h, 311, 313

lib/sensors/MPU6050/MPU6050.cpp, 313  
lib/sensors/MPU6050/MPU6050.h, 314

lib/sensors/MPU6050/MPU6050\_WRAPPER.cpp, 314  
lib/sensors/MPU6050/MPU6050\_WRAPPER.h, 314, 315

lib/storage/storage.cpp, 316, 318  
lib/storage/storage.h, 318, 321

lib/telemetry/telemetry\_manager.cpp, 322, 326  
lib/telemetry/telemetry\_manager.h, 329, 331

lib/utils.cpp, 331, 338  
lib/utils.h, 339, 345

LIGHT  
ISensor.h, 312

LIGHT\_LEVEL  
ISensor.h, 312

light\_level  
TelemetryRecord, 169

LIST\_FILES\_COMMAND  
storage\_commands.cpp, 211

load\_from\_storage  
EventManager, 126  
EventManagerImpl, 130

LOCK  
Event Manager, 48

log\_event  
Event Manager, 48

LOG\_FILENAME  
main.cpp, 347

lon\_dir  
TelemetryRecord, 170

longitude  
TelemetryRecord, 170

LORA  
protocol.h, 228

lora\_address\_local  
pin\_config.cpp, 265  
pin\_config.h, 272

lora\_address\_remote  
pin\_config.cpp, 266  
pin\_config.h, 272

lora\_cs\_pin  
    pin\_config.cpp, 265  
    pin\_config.h, 272  
LORA\_DEFAULT\_DIO0\_PIN  
    pin\_config.h, 271  
LORA\_DEFAULT\_RESET\_PIN  
    pin\_config.h, 271  
LORA\_DEFAULT\_SPI  
    pin\_config.h, 270  
LORA\_DEFAULT\_SPI\_FREQUENCY  
    pin\_config.h, 271  
LORA\_DEFAULT\_SS\_PIN  
    pin\_config.h, 271  
lora\_irq\_pin  
    pin\_config.cpp, 265  
    pin\_config.h, 272  
lora\_reset\_pin  
    pin\_config.cpp, 265  
    pin\_config.h, 272  
lora\_send\_called  
    test\_command\_handlers.cpp, 356  
LOST  
    Event Manager, 48  
LOW\_BATTERY  
    Event Manager, 47  
MAG\_FIELD\_X  
    ISensor.h, 312  
MAG\_FIELD\_Y  
    ISensor.h, 312  
MAG\_FIELD\_Z  
    ISensor.h, 312  
MAGNETOMETER  
    ISensor.h, 312  
main  
    main.cpp, 349  
    test\_runner.cpp, 386  
main.cpp, 346  
    buffer, 350  
    buffer\_index, 350  
    core1\_entry, 347  
    init\_systems, 348  
    LOG\_FILENAME, 347  
    main, 349  
    pause\_gps\_collection, 350  
    powerManager, 350  
    process\_pending\_actions, 347  
    systemClock, 350  
MAIN\_I2C\_PORT  
    pin\_config.h, 268  
MAIN\_I2C\_SCL\_PIN  
    pin\_config.h, 268  
MAIN\_I2C\_SDA\_PIN  
    pin\_config.h, 268  
MAX\_RAW\_DATA\_LENGTH  
    gps\_collector.cpp, 257  
Measurement Functions, 61  
    get\_current\_ma, 62  
    get\_shunt\_voltage, 61  
                get\_voltage, 62  
MILIAMP  
    protocol.h, 228  
minutes  
    ds3231\_data\_t, 118  
mock\_spi\_enabled  
    hardwareMocks.cpp, 373  
    hardwareMocks.h, 375  
mock\_spi\_read\_blocking  
    hardwareMocks.cpp, 372  
    hardwareMocks.h, 375  
mock\_spi\_write\_blocking  
    hardwareMocks.cpp, 372  
    hardwareMocks.h, 375  
mock\_uart\_enabled  
    hardwareMocks.cpp, 373  
    hardwareMocks.h, 375  
mock\_uart\_init  
    hardwareMocks.cpp, 372  
    hardwareMocks.h, 375  
mock\_uart\_puts  
    hardwareMocks.cpp, 372  
    hardwareMocks.h, 375  
Mode  
    BH1750, 75  
mode\_bus\_en  
    INA3221::conf\_reg\_t, 97  
mode\_continious\_en  
    INA3221::conf\_reg\_t, 97  
mode\_shunt\_en  
    INA3221::conf\_reg\_t, 97  
MONDAY  
    DS3231.h, 183  
month  
    ds3231\_data\_t, 119  
MOUNT\_COMMAND  
    storage\_commands.cpp, 211  
MPU6050Wrapper, 151  
    configure, 152  
    get\_type, 152  
    init, 152  
    initialized\_, 153  
    is\_initialized, 152  
    MPU6050Wrapper, 152  
    read\_data, 152  
    sensor\_, 153  
MSG\_RECEIVED  
    Event Manager, 47  
MSG\_SENT  
    Event Manager, 47  
msgCount  
    communication.cpp, 215  
nextEventId  
    EventManager, 127  
nmea\_data  
    gps\_collector.cpp, 258  
    NMEA\_data.cpp, 262  
    NMEA\_data.h, 264

telemetry\_manager.cpp, 325  
**NMEA\_data.cpp**  
 nmea\_data, 262  
**NMEA\_data.h**  
 nmea\_data, 264  
**NMEAData**, 153  
 get\_gga\_tokens, 154  
 get\_rmc\_tokens, 154  
 get\_unix\_time, 155  
 gga\_mutex\_, 156  
 gga\_tokens\_, 155  
 has\_valid\_time, 154  
 NMEAData, 154  
 rmc\_mutex\_, 155  
 rmc\_tokens\_, 155  
 update\_gga\_tokens, 154  
 update\_rmc\_tokens, 154  
**NONE**  
 protocol.h, 227, 228  
**NOT\_ALLOWED**  
 protocol.h, 227, 228  
**NUM\_CALIB\_PARAMS**  
 BME280, 90  
  
**on\_receive**  
 communication.h, 218  
 receive.cpp, 236  
**ONE\_TIME\_HIGH\_RES\_MODE**  
 BH1750, 76  
**ONE\_TIME\_HIGH\_RES\_MODE\_2**  
 BH1750, 76  
**ONE\_TIME\_LOW\_RES\_MODE**  
 BH1750, 76  
**operation\_type\_to\_string**  
 protocol.h, 230  
 utils\_converters.cpp, 244  
**OperationType**  
 protocol.h, 227  
**operationType**  
 Frame, 133  
**outgoing**  
 communication.cpp, 215  
**OVERCHARGE**  
 Event Manager, 47  
  
**PA\_OUTPUT\_PA\_BOOST\_PIN**  
 pin\_config.h, 271  
**PA\_OUTPUT\_RFO\_PIN**  
 pin\_config.h, 271  
**PARAM\_INVALID**  
 protocol.h, 227  
**PARAM\_REQUIRED**  
 protocol.h, 227  
**PARAM\_UNNECESSARY**  
 protocol.h, 228  
**PARAM\_UNNECESSARY**  
 protocol.h, 227  
**PASS\_THROUGH\_END**  
 Event Manager, 48  
  
 PASS\_THROUGH\_START  
 Event Manager, 48  
**PASSTHROUGH\_COMMAND**  
 gps\_commands.cpp, 198  
**pause\_gps\_collection**  
 gps\_collector.cpp, 258  
 main.cpp, 350  
**pin\_config.cpp**  
 lora\_address\_local, 265  
 lora\_address\_remote, 266  
 lora\_cs\_pin, 265  
 lora\_irq\_pin, 265  
 lora\_reset\_pin, 265  
**pin\_config.h**  
 BUFFER\_SIZE, 269  
 DEBUG\_UART\_BAUD\_RATE, 267  
 DEBUG\_UART\_PORT, 267  
 DEBUG\_UART\_RX\_PIN, 268  
 DEBUG\_UART\_TX\_PIN, 268  
 GPS\_POWER\_ENABLE\_PIN, 269  
 GPS\_UART\_BAUD\_RATE, 268  
 GPS\_UART\_PORT, 268  
 GPS\_UART\_RX\_PIN, 269  
 GPS\_UART\_TX\_PIN, 268  
 lora\_address\_local, 272  
 lora\_address\_remote, 272  
 lora\_cs\_pin, 272  
 LORA\_DEFAULT\_DIO0\_PIN, 271  
 LORA\_DEFAULT\_RESET\_PIN, 271  
 LORA\_DEFAULT\_SPI, 270  
 LORA\_DEFAULT\_SPI\_FREQUENCY, 271  
 LORA\_DEFAULT\_SS\_PIN, 271  
 lora\_irq\_pin, 272  
 lora\_reset\_pin, 272  
 MAIN\_I2C\_PORT, 268  
 MAIN\_I2C\_SCL\_PIN, 268  
 MAIN\_I2C\_SDA\_PIN, 268  
 PA\_OUTPUT\_PA\_BOOST\_PIN, 271  
 PA\_OUTPUT\_RFO\_PIN, 271  
 READ\_BIT, 270  
 SD\_CARD\_DETECT\_PIN, 270  
 SD\_CS\_PIN, 269  
 SD\_MISO\_PIN, 269  
 SD\_MOSI\_PIN, 269  
 SD\_SCK\_PIN, 269  
 SD\_SPI\_PORT, 269  
 SPI\_PORT, 270  
 SX1278\_CS, 270  
 SX1278\_MISO, 270  
 SX1278\_MOSI, 270  
 SX1278\_SCK, 270  
**POWER**  
 Event Manager, 46  
**Power Commands**, 28  
 handle\_get\_current\_charge\_solar, 31  
 handle\_get\_current\_charge\_total, 32  
 handle\_get\_current\_charge\_usb, 31  
 handle\_get\_current\_draw, 33

handle\_get\_power\_manager\_ids, 28  
handle\_get\_voltage\_5v, 30  
handle\_get\_voltage\_battery, 29  
power\_commands.cpp  
  CHARGE\_SOLAR, 203  
  CHARGE\_TOTAL, 203  
  CHARGE\_USB, 202  
  DRAW\_TOTAL, 203  
  POWER\_GROUP, 202  
  POWER\_MANAGER\_IDS, 202  
  VOLTAGE\_BATTERY, 202  
  VOLTAGE\_MAIN, 202  
POWER\_FALLING  
  Event Manager, 47  
POWER\_GROUP  
  power\_commands.cpp, 202  
POWER\_MANAGER\_IDS  
  power\_commands.cpp, 202  
POWER\_NORMAL  
  Event Manager, 47  
POWER\_OFF  
  Event Manager, 48  
POWER\_ON  
  BH1750, 76  
  Event Manager, 48  
POWER\_STATUS\_COMMAND  
  gps\_commands.cpp, 198  
PowerEvent  
  Event Manager, 47  
powerman\_mutex\_  
  PowerManager, 161  
PowerManager, 156  
  charging\_solar\_active\_, 161  
  charging\_usb\_active\_, 161  
  check\_power\_alerts, 160  
  configure, 159  
  FALL\_RATE\_THRESHOLD, 161  
  FALLING\_TREND\_REQUIRED, 161  
  get\_current\_charge\_solar, 158  
  get\_current\_charge\_total, 158  
  get\_current\_charge\_usb, 158  
  get\_current\_draw, 159  
  get\_voltage\_5v, 159  
  get\_voltage\_battery, 159  
  ina3221\_, 161  
  initialize, 158  
  initialized\_, 161  
  is\_charging\_solar, 159  
  is\_charging\_usb, 159  
  powerman\_mutex\_, 161  
  PowerManager, 157  
  read\_device\_ids, 158  
  SOLAR\_CURRENT\_THRESHOLD, 160  
  USB\_CURRENT\_THRESHOLD, 160  
  VOLTAGE\_LOW\_THRESHOLD, 160  
  VOLTAGE\_OVERCHARGE\_THRESHOLD, 161  
powerManager  
  main.cpp, 350  
telemetry\_manager.cpp, 324  
PRESSURE  
  ISensor.h, 312  
pressure  
  TelemetryRecord, 169  
process\_pending\_actions  
  main.cpp, 347  
protocol.h  
  BOOL, 228  
  CommandAccessLevel, 227  
  DATETIME, 228  
  DELIMITER, 234  
  ERR, 227  
  error\_code\_to\_string, 229  
  ErrorCode, 226  
  exception\_type\_to\_string, 228  
  ExceptionType, 228  
  FAIL\_TO\_SET, 227  
  FRAME\_BEGIN, 234  
  FRAME\_END, 234  
  GET, 227  
  hex\_string\_to\_bytes, 231  
  Interface, 228  
  INTERNAL\_FAIL\_TO\_READ, 227  
  INVALID\_FORMAT, 227  
  INVALID\_OPERATION, 227, 228  
  INVALID\_PARAM, 228  
  INVALID\_VALUE, 227  
  LORA, 228  
  MILIAMP, 228  
  NONE, 227, 228  
  NOT\_ALLOWED, 227, 228  
  operation\_type\_to\_string, 230  
  OperationType, 227  
  PARAM\_INVALID, 227  
  PARAM\_REQUIRED, 227  
  PARAM\_UNNECESSARY, 228  
  PARAM\_UNNECESSARY, 227  
  READ\_ONLY, 227  
  READ\_WRITE, 227  
  RES, 227  
  SECOND, 228  
  SEQ, 227  
  SET, 227  
  string\_to\_operation\_type, 231  
  TEXT, 228  
  UART, 228  
  UNDEFINED, 228  
  UNKNOWN\_ERROR, 227  
  VAL, 227  
  value\_unit\_type\_to\_string, 232  
  ValueUnit, 227  
  VOLT, 228  
  WRITE\_ONLY, 227  
pwr\_valid\_alert  
  INA3221::masken\_reg\_t, 149  
RADIO\_ERROR  
  Event Manager, 47

RADIO\_INIT  
    Event Manager, 47

read  
    HMC5883L, 135

READ\_BIT  
    pin\_config.h, 270

read\_data  
    BH1750Wrapper, 79  
    BME280Wrapper, 95  
    HMC5883LWrapper, 139  
    ISensor, 147  
    MPU6050Wrapper, 152

read\_device\_ids  
    PowerManager, 158

READ\_ONLY  
    protocol.h, 227

read\_raw\_all  
    BME280, 83

read\_register  
    Configuration Functions, 55  
    HMC5883L, 136

read\_temperature  
    DS3231, 102

READ\_WRITE  
    protocol.h, 227

receive.cpp  
    handle\_uart\_input, 237  
    on\_receive, 236

REG\_CONFIG  
    BME280, 85

REG\_CTRL\_HUM  
    BME280, 86

REG\_CTRL\_MEAS  
    BME280, 86

REG\_DIG\_H1  
    BME280, 89

REG\_DIG\_H2  
    BME280, 89

REG\_DIG\_H3  
    BME280, 90

REG\_DIG\_H4  
    BME280, 90

REG\_DIG\_H5  
    BME280, 90

REG\_DIG\_H6  
    BME280, 90

REG\_DIG\_P1\_LSB  
    BME280, 87

REG\_DIG\_P1\_MSB  
    BME280, 87

REG\_DIG\_P2\_LSB  
    BME280, 87

REG\_DIG\_P2\_MSB  
    BME280, 87

REG\_DIG\_P3\_LSB  
    BME280, 88

REG\_DIG\_P3\_MSB  
    BME280, 88

REG\_DIG\_P4\_LSB  
    BME280, 88

REG\_DIG\_P4\_MSB  
    BME280, 88

REG\_DIG\_P5\_LSB  
    BME280, 88

REG\_DIG\_P5\_MSB  
    BME280, 88

REG\_DIG\_P6\_LSB  
    BME280, 88

REG\_DIG\_P6\_MSB  
    BME280, 88

REG\_DIG\_P7\_LSB  
    BME280, 89

REG\_DIG\_P7\_MSB  
    BME280, 89

REG\_DIG\_P8\_LSB  
    BME280, 89

REG\_DIG\_P8\_MSB  
    BME280, 89

REG\_DIG\_P9\_LSB  
    BME280, 89

REG\_DIG\_P9\_MSB  
    BME280, 89

REG\_DIG\_T1\_LSB  
    BME280, 86

REG\_DIG\_T1\_MSB  
    BME280, 86

REG\_DIG\_T2\_LSB  
    BME280, 87

REG\_DIG\_T2\_MSB  
    BME280, 87

REG\_DIG\_T3\_LSB  
    BME280, 87

REG\_DIG\_T3\_MSB  
    BME280, 87

REG\_HUMIDITY\_MSB  
    BME280, 86

REG\_PRESSURE\_MSB  
    BME280, 86

REG\_RESET  
    BME280, 86

REG\_TEMPERATURE\_MSB  
    BME280, 86

RES  
    protocol.h, 227

reserved  
    INA3221::masken\_reg\_t, 150

RESET  
    BH1750, 76

reset  
    BME280, 83  
    Configuration Functions, 53  
    INA3221::conf\_reg\_t, 98

RMC\_DATA\_COMMAND  
    gps\_commands.cpp, 198

rmc\_mutex\_  
    NMEAData, 155

rmc\_tokens\_  
    NMEAData, 155

sample\_interval\_ms  
    telemetry\_manager.cpp, 325

satellites  
    TelemetryRecord, 171

SATURDAY  
    DS3231.h, 183

save\_to\_storage  
    EventManager, 125  
    EventManagerImpl, 130

scan\_connected\_sensors  
    SensorWrapper, 166

SD\_CARD\_DETECT\_PIN  
    pin\_config.h, 270

sd\_card\_mounted  
    storage.cpp, 318  
    storage.h, 321

SD\_CS\_PIN  
    pin\_config.h, 269

SD\_MISO\_PIN  
    pin\_config.h, 269

SD\_MOSI\_PIN  
    pin\_config.h, 269

SD\_SCK\_PIN  
    pin\_config.h, 269

SD\_SPI\_PORT  
    pin\_config.h, 269

SECOND  
    protocol.h, 228

seconds  
    ds3231\_data\_t, 118

send.cpp  
    send\_frame\_lora, 240  
    send\_frame\_uart, 240  
    send\_message, 239  
    split\_and\_send\_message, 240

send\_frame\_lora  
    communication.h, 220  
    send.cpp, 240  
    test\_command\_handlers.cpp, 354

send\_frame\_uart  
    communication.h, 219  
    send.cpp, 240  
    test\_command\_handlers.cpp, 353

send\_message  
    communication.h, 219  
    send.cpp, 239

Sensor Commands, 34  
    handle\_get\_sensor\_data, 34  
    handle\_get\_sensor\_list, 36  
    handle\_sensor\_config, 35

sensor\_  
    BH1750Wrapper, 80  
    BME280Wrapper, 96  
    HMC5883LWrapper, 140  
    MPU6050Wrapper, 153

sensor\_commands.cpp

    SENSOR\_CONFIGURE, 206  
    SENSOR\_GROUP, 206  
    SENSOR\_READ, 206

    SENSOR\_CONFIGURE  
        sensor\_commands.cpp, 206

    sensor\_configure  
        SensorWrapper, 164

    SENSOR\_GROUP  
        sensor\_commands.cpp, 206

    sensor\_init  
        SensorWrapper, 164

    SENSOR\_READ  
        sensor\_commands.cpp, 206

    sensor\_read\_data  
        SensorWrapper, 165

    SensorDataTypIdentifier  
        ISensor.h, 312

    sensors  
        SensorWrapper, 167

    SensorType  
        ISensor.h, 312

    SensorWrapper, 162  
        get\_available\_sensors, 166  
        get\_instance, 163  
        get\_sensor, 165  
        scan\_connected\_sensors, 166  
        sensor\_configure, 164  
        sensor\_init, 164  
        sensor\_read\_data, 165  
        sensors, 167  
        SensorWrapper, 163

SEQ  
    protocol.h, 227

SET  
    protocol.h, 227

set\_alert\_latch  
    Alert Functions, 68

set\_averaging\_mode  
    Configuration Functions, 59

set\_bus\_conversion\_time  
    Configuration Functions, 59

set\_bus\_measurement\_disable  
    Configuration Functions, 58

set\_bus\_measurement\_enable  
    Configuration Functions, 58

set\_clock\_sync\_interval  
    DS3231, 108

set\_crit\_alert\_limit  
    Alert Functions, 64

set\_mode\_continuous  
    Configuration Functions, 56

set\_mode\_power\_down  
    Configuration Functions, 55

set\_mode\_triggered  
    Configuration Functions, 56

set\_power\_valid\_limit  
    Alert Functions, 65

set\_shunt\_conversion\_time

Configuration Functions, 60  
**set\_shunt\_measurement\_disable**  
 Configuration Functions, 57  
**set\_shunt\_measurement\_enable**  
 Configuration Functions, 57  
**set\_telemetry\_flush\_threshold**  
 Telemetry Manager, 74  
**set\_telemetry\_sample\_interval**  
 Telemetry Manager, 74  
**set\_time**  
 DS3231, 100  
**set\_timezone\_offset**  
 DS3231, 107  
**set\_unix\_time**  
 DS3231, 104  
**set\_warn\_alert\_limit**  
 Alert Functions, 64  
**setUp**  
 test\_comand\_handlers.cpp, 354  
 test\_error\_codes.cpp, 361  
 test\_frame\_send.cpp, 370  
**shunt\_conv\_time**  
 INA3221::conf\_reg\_t, 97  
**shunt\_sum\_alert**  
 INA3221::masken\_reg\_t, 149  
**shunt\_sum\_en\_ch1**  
 INA3221::masken\_reg\_t, 150  
**shunt\_sum\_en\_ch2**  
 INA3221::masken\_reg\_t, 150  
**shunt\_sum\_en\_ch3**  
 INA3221::masken\_reg\_t, 150  
**SHUNT\_VOLTAGE\_LSB\_UV**  
 INA3221.h, 282  
**SHUTDOWN**  
 Event Manager, 47  
**SILENT**  
 utils.h, 342  
**SOLAR\_ACTIVE**  
 Event Manager, 47  
**SOLAR\_CURRENT\_THRESHOLD**  
 PowerManager, 160  
**SOLAR\_INACTIVE**  
 Event Manager, 47  
**speed**  
 TelemetryRecord, 170  
**spi\_output\_buffer**  
 hardwareMocks.cpp, 373  
 hardwareMocks.h, 376  
**SPI\_PORT**  
 pin\_config.h, 270  
**split\_and\_send\_message**  
 communication.h, 221  
 send.cpp, 240  
**splitString**  
 gpsCollector.cpp, 257  
**Storage Commands**, 37  
 handle\_list\_files, 38  
 handle\_mount, 38  
**storage.cpp**  
 fs\_init, 317  
 sd\_card\_mounted, 318  
**storage.h**  
 fs\_close\_file, 321  
 fs\_file\_exists, 321  
 fs\_init, 319  
 fs\_open\_file, 320  
 fs\_read\_file, 320  
 fs\_write\_file, 320  
 sd\_card\_mounted, 321  
**storage\_commands.cpp**  
 LIST\_FILES\_COMMAND, 211  
 MOUNT\_COMMAND, 211  
 STORAGE\_GROUP, 211  
**STORAGE\_GROUP**  
 storage\_commands.cpp, 211  
**string\_to\_operation\_type**  
 protocol.h, 231  
 utils\_converters.cpp, 245  
**SUNDAY**  
 DS3231.h, 183  
**SX1278\_CS**  
 pin\_config.h, 270  
**SX1278\_MISO**  
 pin\_config.h, 270  
**SX1278\_MOSI**  
 pin\_config.h, 270  
**SX1278\_SCK**  
 pin\_config.h, 270  
**sync\_clock\_with\_gps**  
 DS3231, 111  
**sync\_interval\_minutes\_**  
 DS3231, 117  
**SYSTEM**  
 Event Manager, 46  
**system\_voltage**  
 TelemetryRecord, 168  
**systemClock**  
 Clock Management Commands, 15  
 Event Manager, 50  
 main.cpp, 350  
 telemetry\_manager.cpp, 324  
**SystemEvent**  
 Event Manager, 46  
**t\_fine**  
 BME280, 85  
**tearDown**  
 test\_comand\_handlers.cpp, 354  
 test\_error\_codes.cpp, 361  
 test\_frame\_send.cpp, 370  
**Telemetry Manager**, 69  
 collect\_telemetry, 70  
 flush\_telemetry, 71  
 get\_telemetry\_flush\_threshold, 74  
 get\_telemetry\_sample\_interval, 73  
 is\_telemetry\_collection\_time, 72  
 is\_telemetry\_flush\_time, 73

set\_telemetry\_flush\_threshold, 74  
set\_telemetry\_sample\_interval, 74  
telemetry\_init, 70  
telemetry\_buffer  
    telemetry\_manager.cpp, 325  
telemetry\_buffer\_count  
    telemetry\_manager.cpp, 325  
TELEMETRY\_BUFFER\_SIZE  
    telemetry\_manager.cpp, 324  
telemetry\_buffer\_write\_index  
    telemetry\_manager.cpp, 325  
TELEMETRY\_CSV\_PATH  
    telemetry\_manager.cpp, 324  
telemetry\_init  
    Telemetry Manager, 70  
telemetry\_manager.cpp  
    DEFAULT\_FLUSH\_THRESHOLD, 324  
    DEFAULT\_SAMPLE\_INTERVAL\_MS, 324  
    flush\_threshold, 325  
    nmea\_data, 325  
    powerManager, 324  
    sample\_interval\_ms, 325  
    systemClock, 324  
    telemetry\_buffer, 325  
    telemetry\_buffer\_count, 325  
    TELEMETRY\_BUFFER\_SIZE, 324  
    telemetry\_buffer\_write\_index, 325  
    TELEMETRY\_CSV\_PATH, 324  
    telemetry\_mutex, 325  
telemetry\_mutex  
    telemetry\_manager.cpp, 325  
TelemetryRecord, 167  
    altitude, 171  
    battery\_voltage, 168  
    charge\_current\_solar, 168  
    charge\_current\_usb, 168  
    course, 170  
    date, 171  
    discharge\_current, 169  
    fix\_quality, 171  
    humidity, 169  
    lat\_dir, 170  
    latitude, 170  
    light\_level, 169  
    lon\_dir, 170  
    longitude, 170  
    pressure, 169  
    satellites, 171  
    speed, 170  
    system\_voltage, 168  
    temperature, 169  
    time, 169  
    timestamp, 168  
    to\_csv, 168  
TEMPERATURE  
    ISensor.h, 312  
temperature  
    TelemetryRecord, 169  
  
test/comms/test\_comand\_handlers.cpp, 352, 357  
test/comms/test\_converters.cpp, 358, 360  
test/comms/test\_error\_codes.cpp, 361, 362  
test/comms/test\_frame\_build.cpp, 363, 365  
test/comms/test\_frame\_coding.cpp, 365, 367  
test/comms/test\_frame\_common.h, 368, 369  
test/comms/test\_frame\_send.cpp, 370, 371  
test/mocks/hardwareMocks.cpp, 371, 373  
test/mocks/hardwareMocks.h, 374, 376  
test/test\_runner.cpp, 376, 386  
test\_comand\_handlers.cpp  
    last\_frame\_sent, 357  
    lora\_send\_called, 356  
    send\_frame\_lora, 354  
    send\_frame\_uart, 353  
    setUp, 354  
    tearDown, 354  
    test\_command\_handler\_get\_operation, 355  
    test\_command\_handler\_invalid\_operation, 356  
    test\_command\_handler\_set\_operation, 355  
    uart\_send\_called, 356  
test\_command\_handler\_get\_operation  
    test\_comand\_handlers.cpp, 355  
    test\_runner.cpp, 383  
test\_command\_handler\_invalid\_operation  
    test\_comand\_handlers.cpp, 356  
    test\_runner.cpp, 384  
test\_command\_handler\_set\_operation  
    test\_comand\_handlers.cpp, 355  
    test\_runner.cpp, 384  
test\_converters.cpp  
    test\_exception\_type\_conversion, 359  
    test\_hex\_string\_conversion, 360  
    test\_operation\_type\_conversion, 358  
    test\_value\_unit\_type\_conversion, 358  
test\_error\_code\_conversion  
    test\_error\_codes.cpp, 362  
    test\_runner.cpp, 385  
test\_error\_codes.cpp  
    setUp, 361  
    tearDown, 361  
    test\_error\_code\_conversion, 362  
test\_exception\_type\_conversion  
    test\_converters.cpp, 359  
    test\_runner.cpp, 382  
test\_frame\_build.cpp  
    test\_frame\_build\_error, 363  
    test\_frame\_build\_info, 364  
    test\_frame\_build\_success, 363  
test\_frame\_build\_error  
    test\_frame\_build.cpp, 363  
    test\_runner.cpp, 379  
test\_frame\_build\_info  
    test\_frame\_build.cpp, 364  
    test\_runner.cpp, 380  
test\_frame\_build\_success  
    test\_frame\_build.cpp, 363  
    test\_runner.cpp, 379

test\_frame\_coding.cpp  
     test\_frame\_decode\_basic, 366  
     test\_frame\_decode\_invalid\_header, 367  
     test\_frame\_encode\_basic, 366

test\_frame\_common.h  
     create\_test\_frame, 369

test\_frame\_decode\_basic  
     test\_frame\_coding.cpp, 366  
     test\_runner.cpp, 377

test\_frame\_decode\_invalid\_header  
     test\_frame\_coding.cpp, 367  
     test\_runner.cpp, 378

test\_frame\_encode\_basic  
     test\_frame\_coding.cpp, 366  
     test\_runner.cpp, 377

test\_frame\_send.cpp  
     setUp, 370  
     tearDown, 370  
     test\_send\_frame\_uart, 370

test\_hex\_string\_conversion  
     test\_converters.cpp, 360  
     test\_runner.cpp, 382

test\_operation\_type\_conversion  
     test\_converters.cpp, 358  
     test\_runner.cpp, 380

test\_runner.cpp  
     main, 386  
     test\_command\_handler\_get\_operation, 383  
     test\_command\_handler\_invalid\_operation, 384  
     test\_command\_handler\_set\_operation, 384  
     test\_error\_code\_conversion, 385  
     test\_exception\_type\_conversion, 382  
     test\_frame\_build\_error, 379  
     test\_frame\_build\_info, 380  
     test\_frame\_build\_success, 379  
     test\_frame\_decode\_basic, 377  
     test\_frame\_decode\_invalid\_header, 378  
     test\_frame\_encode\_basic, 377  
     test\_hex\_string\_conversion, 382  
     test\_operation\_type\_conversion, 380  
     test\_value\_unit\_type\_conversion, 381

test\_send\_frame\_uart  
     test\_frame\_send.cpp, 370

test\_value\_unit\_type\_conversion  
     test\_converters.cpp, 358  
     test\_runner.cpp, 381

TEXT  
     protocol.h, 228

THURSDAY  
     DS3231.h, 183

TIME  
     clock\_commands.cpp, 186

time  
     TelemetryRecord, 169

timestamp  
     event\_manager.h, 253  
     EventLog, 122  
     TelemetryRecord, 168

TIMEZONE\_OFFSET  
     clock\_commands.cpp, 186

timezone\_offset\_minutes\_  
     DS3231, 117

timing\_ctrl\_alert  
     INA3221::masken\_reg\_t, 149

to\_csv  
     TelemetryRecord, 168

to\_string  
     event\_manager.h, 253  
     EventLog, 122

TUESDAY  
     DS3231.h, 183

UART  
     protocol.h, 228

UART\_ERROR  
     Event Manager, 47

uart\_mutex  
     utils.cpp, 337

uart\_output\_buffer  
     hardwareMocks.cpp, 373  
     hardwareMocks.h, 375

uart\_print  
     utils.cpp, 334  
     utils.h, 342

uart\_send\_called  
     test\_comand\_handlers.cpp, 356

UNCONFIGURED\_POWER\_DOWN  
     BH1750, 76

UNDEFINED  
     protocol.h, 228

unit  
     Frame, 134

UNKNOWN\_ERROR  
     protocol.h, 227

update\_gga\_tokens  
     NMEAData, 154

update\_last\_sync\_time  
     DS3231, 109

update\_rmc\_tokens  
     NMEAData, 154

USB\_CONNECTED  
     Event Manager, 47

USB\_CURRENT\_THRESHOLD  
     PowerManager, 160

USB\_DISCONNECTED  
     Event Manager, 47

utils.cpp  
     base64\_chars, 337  
     base64\_encode, 336  
     calculate\_checksum, 336  
     g\_uart\_verbosity, 337  
     get\_level\_color, 332  
     get\_level\_prefix, 333  
     uart\_mutex, 337  
     uart\_print, 334

utils.h  
     ANSI\_BLUE, 341

ANSI\_CYAN, 341  
ANSI\_GREEN, 341  
ANSI\_RED, 341  
ANSI\_RESET, 341  
ANSI\_YELLOW, 341  
base64\_encode, 344  
calculate\_checksum, 344  
DEBUG, 342  
ERROR, 342  
EVENT, 342  
g\_uart\_verbosity, 345  
INFO, 342  
SILENT, 342  
uart\_print, 342  
VerbosityLevel, 341  
WARNING, 342  
utils\_converters.cpp  
    error\_code\_to\_string, 245  
    exception\_type\_to\_string, 242  
    hex\_string\_to\_bytes, 247  
    operation\_type\_to\_string, 244  
    string\_to\_operation\_type, 245  
    value\_unit\_type\_to\_string, 243

VAL  
    protocol.h, 227

value  
    Frame, 134

value\_unit\_type\_to\_string  
    protocol.h, 232  
    utils\_converters.cpp, 243

ValueUnit  
    protocol.h, 227

VerbosityLevel  
    utils.h, 341

VOLT  
    protocol.h, 228

VOLTAGE\_BATTERY  
    power\_commands.cpp, 202

VOLTAGE\_LOW\_THRESHOLD  
    PowerManager, 160

VOLTAGE\_MAIN  
    power\_commands.cpp, 202

VOLTAGE\_OVERCHARGE\_THRESHOLD  
    PowerManager, 161

warn\_alert\_ch1  
    INA3221::masken\_reg\_t, 149

warn\_alert\_ch2  
    INA3221::masken\_reg\_t, 149

warn\_alert\_ch3  
    INA3221::masken\_reg\_t, 149

warn\_alert\_latch\_en  
    INA3221::masken\_reg\_t, 150

WARNING  
    utils.h, 342

WATCHDOG\_RESET  
    Event Manager, 47

WEDNESDAY