

KubiSat Firmware

Generated by Doxygen 1.13.2

1 Clock Commands	1
2 Topic Index	3
2.1 Topics	3
3 Hierarchical Index	5
3.1 Class Hierarchy	5
4 Class Index	7
4.1 Class List	7
5 File Index	9
5.1 File List	9
6 Topic Documentation	11
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	14
6.1.3.1 systemClock	14
6.2 Command System	14
6.2.1 Detailed Description	15
6.2.2 Typedef Documentation	15
6.2.2.1 CommandHandler	15
6.2.2.2 CommandMap	15
6.2.3 Function Documentation	15
6.2.3.1 execute_command()	15
6.2.4 Variable Documentation	16
6.2.4.1 commandHandlers	16
6.3 Diagnostic Commands	17
6.3.1 Detailed Description	17
6.3.2 Function Documentation	17
6.3.2.1 handle_get_commands_list()	17
6.3.2.2 handle_get_build_version()	18
6.3.2.3 handle_verbosity()	19
6.3.2.4 handle_enter_bootloader_mode()	20
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	21

6.4.2 Function Documentation	21
6.4.2.1 handle_get_last_events()	21
6.4.2.2 handle_get_event_count()	22
6.5 GPS Commands	23
6.5.1 Detailed Description	23
6.5.2 Function Documentation	23
6.5.2.1 handle_gps_power_status()	23
6.5.2.2 handle_enable_gps_uart_passthrough()	24
6.5.2.3 handle_get_rmc_data()	25
6.5.2.4 handle_get_gga_data()	26
6.6 Power Commands	27
6.6.1 Detailed Description	27
6.6.2 Function Documentation	28
6.6.2.1 handle_get_power_manager_ids()	28
6.6.2.2 handle_get_voltage_battery()	28
6.6.2.3 handle_get_voltage_5v()	29
6.6.2.4 handle_get_current_charge_usb()	30
6.6.2.5 handle_get_current_charge_solar()	31
6.6.2.6 handle_get_current_charge_total()	31
6.6.2.7 handle_get_current_draw()	32
6.7 Storage Commands	33
6.7.1 Detailed Description	33
6.7.2 Function Documentation	33
6.7.2.1 handle_file_download()	33
6.7.2.2 handle_list_files()	34
6.7.2.3 handle_mount()	35
6.8 Frame Handling	36
6.8.1 Detailed Description	36
6.8.2 Function Documentation	36
6.8.2.1 frame_encode()	36
6.8.2.2 frame_decode()	37
6.8.2.3 frame_process()	38
6.8.2.4 frame_build()	39
6.9 Event Manager	41
6.9.1 Detailed Description	43
6.9.2 Enumeration Type Documentation	43
6.9.2.1 EventGroup	43
6.9.2.2 SystemEvent	44
6.9.2.3 PowerEvent	44
6.9.2.4 CommsEvent	44
6.9.2.5 GPSEvent	45
6.9.2.6 ClockEvent	45

6.9.3 Function Documentation	45
6.9.3.1 check_power_events()	45
6.9.3.2 __attribute__()	46
6.9.3.3 log_event()	46
6.9.3.4 get_event()	47
6.9.4 Variable Documentation	48
6.9.4.1 eventLogId	48
6.9.4.2 lastPowerState	48
6.9.4.3 FALL_RATE_THRESHOLD	48
6.9.4.4 FALLING_TREND_REQUIRED	48
6.9.4.5 VOLTAGE_LOW_THRESHOLD	48
6.9.4.6 VOLTAGE_OVERCHARGE_THRESHOLD	48
6.9.4.7 fallingTrendCount	49
6.9.4.8 lastSolarState	49
6.9.4.9 lastUSBState	49
6.9.4.10 systemClock	49
6.9.4.11 eventManager [1/2]	49
6.9.4.12 __attribute__	49
6.9.4.13 eventManager [2/2]	50
6.10 INA3221 Power Monitor	50
6.10.1 Detailed Description	50
6.10.2 Configuration Functions	50
6.10.2.1 Detailed Description	51
6.10.2.2 Function Documentation	51
6.10.3 Measurement Functions	60
6.10.3.1 Detailed Description	60
6.10.3.2 Function Documentation	60
6.10.4 Alert Functions	62
6.10.4.1 Detailed Description	63
6.10.4.2 Function Documentation	63
7 Class Documentation	69
7.1 BH1750 Class Reference	69
7.1.1 Detailed Description	69
7.1.2 Member Enumeration Documentation	69
7.1.2.1 Mode	69
7.1.3 Constructor & Destructor Documentation	70
7.1.3.1 BH1750()	70
7.1.4 Member Function Documentation	70
7.1.4.1 begin()	70
7.1.4.2 configure()	71
7.1.4.3 get_light_level()	71

7.1.4.4 write8()	71
7.1.5 Member Data Documentation	72
7.1.5.1 _i2c_addr	72
7.2 BH1750Wrapper Class Reference	72
7.2.1 Detailed Description	73
7.2.2 Constructor & Destructor Documentation	73
7.2.2.1 BH1750Wrapper()	73
7.2.3 Member Function Documentation	73
7.2.3.1 get_i2c_addr()	73
7.2.3.2 init()	73
7.2.3.3 read_data()	74
7.2.3.4 is_initialized()	74
7.2.3.5 get_type()	74
7.2.3.6 configure()	74
7.2.4 Member Data Documentation	74
7.2.4.1 sensor_	74
7.2.4.2 initialized_	74
7.3 BME280 Class Reference	75
7.3.1 Detailed Description	76
7.3.2 Constructor & Destructor Documentation	76
7.3.2.1 BME280()	76
7.3.3 Member Function Documentation	77
7.3.3.1 init()	77
7.3.3.2 reset()	77
7.3.3.3 read_raw_all()	77
7.3.3.4 convert_temperature()	77
7.3.3.5 convert_pressure()	77
7.3.3.6 convert_humidity()	78
7.3.3.7 configure_sensor()	78
7.3.3.8 get_calibration_parameters()	78
7.3.4 Member Data Documentation	78
7.3.4.1 ADDR_SDO_LOW	78
7.3.4.2 ADDR_SDO_HIGH	79
7.3.4.3 i2c_port	79
7.3.4.4 device_addr	79
7.3.4.5 calib_params	79
7.3.4.6 initialized_	79
7.3.4.7 t_fine	79
7.3.4.8 REG_CONFIG	79
7.3.4.9 REG_CTRL_MEAS	79
7.3.4.10 REG_CTRL_HUM	80
7.3.4.11 REG_RESET	80

7.3.4.12 REG_PRESSURE_MSB	80
7.3.4.13 REG_TEMPERATURE_MSB	80
7.3.4.14 REG_HUMIDITY_MSB	80
7.3.4.15 REG_DIG_T1_LSB	80
7.3.4.16 REG_DIG_T1_MSB	80
7.3.4.17 REG_DIG_T2_LSB	80
7.3.4.18 REG_DIG_T2_MSB	81
7.3.4.19 REG_DIG_T3_LSB	81
7.3.4.20 REG_DIG_T3_MSB	81
7.3.4.21 REG_DIG_P1_LSB	81
7.3.4.22 REG_DIG_P1_MSB	81
7.3.4.23 REG_DIG_P2_LSB	81
7.3.4.24 REG_DIG_P2_MSB	81
7.3.4.25 REG_DIG_P3_LSB	81
7.3.4.26 REG_DIG_P3_MSB	82
7.3.4.27 REG_DIG_P4_LSB	82
7.3.4.28 REG_DIG_P4_MSB	82
7.3.4.29 REG_DIG_P5_LSB	82
7.3.4.30 REG_DIG_P5_MSB	82
7.3.4.31 REG_DIG_P6_LSB	82
7.3.4.32 REG_DIG_P6_MSB	82
7.3.4.33 REG_DIG_P7_LSB	82
7.3.4.34 REG_DIG_P7_MSB	83
7.3.4.35 REG_DIG_P8_LSB	83
7.3.4.36 REG_DIG_P8_MSB	83
7.3.4.37 REG_DIG_P9_LSB	83
7.3.4.38 REG_DIG_P9_MSB	83
7.3.4.39 REG_DIG_H1	83
7.3.4.40 REG_DIG_H2	83
7.3.4.41 REG_DIG_H3	83
7.3.4.42 REG_DIG_H4	84
7.3.4.43 REG_DIG_H5	84
7.3.4.44 REG_DIG_H6	84
7.3.4.45 NUM_CALIB_PARAMS	84
7.4 BME280CalibParam Struct Reference	84
7.4.1 Detailed Description	85
7.4.2 Member Data Documentation	85
7.4.2.1 dig_t1	85
7.4.2.2 dig_t2	85
7.4.2.3 dig_t3	85
7.4.2.4 dig_p1	85
7.4.2.5 dig_p2	85

7.4.2.6 dig_p3	85
7.4.2.7 dig_p4	86
7.4.2.8 dig_p5	86
7.4.2.9 dig_p6	86
7.4.2.10 dig_p7	86
7.4.2.11 dig_p8	86
7.4.2.12 dig_p9	86
7.4.2.13 dig_h1	86
7.4.2.14 dig_h2	86
7.4.2.15 dig_h3	87
7.4.2.16 dig_h4	87
7.4.2.17 dig_h5	87
7.4.2.18 dig_h6	87
7.5 BME280Wrapper Class Reference	87
7.5.1 Detailed Description	88
7.5.2 Constructor & Destructor Documentation	88
7.5.2.1 BME280Wrapper()	88
7.5.3 Member Function Documentation	89
7.5.3.1 init()	89
7.5.3.2 read_data()	89
7.5.3.3 is_initialized()	89
7.5.3.4 get_type()	89
7.5.3.5 configure()	89
7.5.4 Member Data Documentation	89
7.5.4.1 sensor_	89
7.5.4.2 initialized_	90
7.6 INA3221::conf_reg_t Struct Reference	90
7.6.1 Detailed Description	90
7.6.2 Member Data Documentation	90
7.6.2.1 mode_shunt_en	90
7.6.2.2 mode_bus_en	90
7.6.2.3 mode_continious_en	91
7.6.2.4 shunt_conv_time	91
7.6.2.5 bus_conv_time	91
7.6.2.6 avg_mode	91
7.6.2.7 ch3_en	91
7.6.2.8 ch2_en	91
7.6.2.9 ch1_en	91
7.6.2.10 reset	92
7.7 DS3231 Class Reference	92
7.7.1 Detailed Description	92
7.7.2 Constructor & Destructor Documentation	93

7.7.2.1 DS3231()	93
7.7.3 Member Function Documentation	93
7.7.3.1 set_time()	93
7.7.3.2 get_time()	94
7.7.3.3 read_temperature()	94
7.7.3.4 set_unix_time()	95
7.7.3.5 get_unix_time()	95
7.7.3.6 clock_enable()	95
7.7.3.7 i2c_read_reg()	96
7.7.3.8 i2c_write_reg()	97
7.7.3.9 bin_to_bcd()	97
7.7.3.10 bcd_to_bin()	98
7.7.4 Member Data Documentation	99
7.7.4.1 i2c	99
7.7.4.2 ds3231_addr	99
7.7.4.3 clock_mutex_	99
7.8 ds3231_data_t Struct Reference	99
7.8.1 Detailed Description	99
7.8.2 Member Data Documentation	100
7.8.2.1 seconds	100
7.8.2.2 minutes	100
7.8.2.3 hours	100
7.8.2.4 day	100
7.8.2.5 date	100
7.8.2.6 month	100
7.8.2.7 year	100
7.8.2.8 century	101
7.9 EventEmitter Class Reference	101
7.9.1 Detailed Description	101
7.9.2 Member Function Documentation	101
7.9.2.1 emit()	101
7.10 EventLog Class Reference	102
7.10.1 Detailed Description	103
7.10.2 Member Function Documentation	103
7.10.2.1 to_string()	103
7.10.3 Member Data Documentation	103
7.10.3.1 id	103
7.10.3.2 timestamp	104
7.10.3.3 group	104
7.10.3.4 event	104
7.11 EventManager Class Reference	104
7.11.1 Detailed Description	106

7.11.2 Constructor & Destructor Documentation	106
7.11.2.1 EventManager()	106
7.11.2.2 ~EventManager()	106
7.11.3 Member Function Documentation	106
7.11.3.1 init()	106
7.11.3.2 get_event_count()	107
7.11.3.3 save_to_storage()	107
7.11.3.4 load_from_storage()	107
7.11.4 Member Data Documentation	108
7.11.4.1 events	108
7.11.4.2 eventCount	108
7.11.4.3 writeIndex	108
7.11.4.4 eventMutex	108
7.11.4.5 nextEventId	108
7.11.4.6 needsPersistence	108
7.12 EventManagerImpl Class Reference	109
7.12.1 Detailed Description	110
7.12.2 Constructor & Destructor Documentation	110
7.12.2.1 EventManagerImpl()	110
7.12.3 Member Function Documentation	111
7.12.3.1 save_to_storage()	111
7.12.3.2 load_from_storage()	111
7.13 FileHandle Struct Reference	112
7.13.1 Detailed Description	112
7.13.2 Member Data Documentation	112
7.13.2.1 fd	112
7.13.2.2 is_open	112
7.14 Frame Struct Reference	112
7.14.1 Detailed Description	113
7.14.2 Member Data Documentation	114
7.14.2.1 header	114
7.14.2.2 direction	114
7.14.2.3 operationType	114
7.14.2.4 group	114
7.14.2.5 command	114
7.14.2.6 value	114
7.14.2.7 unit	114
7.14.2.8 footer	115
7.15 HMC5883L Class Reference	115
7.15.1 Detailed Description	115
7.15.2 Constructor & Destructor Documentation	115
7.15.2.1 HMC5883L()	115

7.15.3 Member Function Documentation	116
7.15.3.1 init()	116
7.15.3.2 read()	116
7.15.3.3 write_register()	116
7.15.3.4 read_register()	117
7.15.4 Member Data Documentation	117
7.15.4.1 i2c	117
7.15.4.2 address	117
7.16 HMC5883LWrapper Class Reference	118
7.16.1 Detailed Description	119
7.16.2 Constructor & Destructor Documentation	119
7.16.2.1 HMC5883LWrapper()	119
7.16.3 Member Function Documentation	119
7.16.3.1 init()	119
7.16.3.2 read_data()	119
7.16.3.3 is_initialized()	119
7.16.3.4 get_type()	120
7.16.3.5 configure()	120
7.16.4 Member Data Documentation	120
7.16.4.1 sensor_	120
7.16.4.2 initialized_	120
7.17 INA3221 Class Reference	120
7.17.1 Detailed Description	122
7.17.2 Member Function Documentation	122
7.17.2.1 _read()	122
7.17.2.2 _write()	124
7.17.2.3 get_current()	125
7.17.3 Member Data Documentation	126
7.17.3.1 _i2c	126
7.17.3.2 _i2c_addr	126
7.17.3.3 _shuntRes	126
7.17.3.4 _filterRes	126
7.17.3.5 _masken_reg	126
7.18 ISensor Class Reference	126
7.18.1 Detailed Description	127
7.18.2 Constructor & Destructor Documentation	127
7.18.2.1 ~ISensor()	127
7.18.3 Member Function Documentation	127
7.18.3.1 init()	127
7.18.3.2 read_data()	127
7.18.3.3 is_initialized()	127
7.18.3.4 get_type()	127

7.18.3.5 configure()	128
7.19 INA3221::masken_reg_t Struct Reference	128
7.19.1 Detailed Description	128
7.19.2 Member Data Documentation	128
7.19.2.1 conv_ready	128
7.19.2.2 timing_ctrl_alert	129
7.19.2.3 pwr_valid_alert	129
7.19.2.4 warn_alert_ch3	129
7.19.2.5 warn_alert_ch2	129
7.19.2.6 warn_alert_ch1	129
7.19.2.7 shunt_sum_alert	129
7.19.2.8 crit_alert_ch3	129
7.19.2.9 crit_alert_ch2	129
7.19.2.10 crit_alert_ch1	130
7.19.2.11 crit_alert_latch_en	130
7.19.2.12 warn_alert_latch_en	130
7.19.2.13 shunt_sum_en_ch3	130
7.19.2.14 shunt_sum_en_ch2	130
7.19.2.15 shunt_sum_en_ch1	130
7.19.2.16 reserved	130
7.20 MPU6050Wrapper Class Reference	131
7.20.1 Detailed Description	132
7.20.2 Constructor & Destructor Documentation	132
7.20.2.1 MPU6050Wrapper()	132
7.20.3 Member Function Documentation	132
7.20.3.1 init()	132
7.20.3.2 read_data()	132
7.20.3.3 is_initialized()	132
7.20.3.4 get_type()	132
7.20.3.5 configure()	133
7.20.4 Member Data Documentation	133
7.20.4.1 sensor_	133
7.20.4.2 initialized_	133
7.21 NMEAData Class Reference	133
7.21.1 Detailed Description	133
7.21.2 Constructor & Destructor Documentation	134
7.21.2.1 NMEAData()	134
7.21.3 Member Function Documentation	134
7.21.3.1 update_rmc_tokens()	134
7.21.3.2 update_gga_tokens()	134
7.21.3.3 get_rmc_tokens()	134
7.21.3.4 get_gga_tokens()	134

7.21.4 Member Data Documentation	134
7.21.4.1 rmc_tokens_	134
7.21.4.2 gga_tokens_	135
7.21.4.3 rmc_mutex_	135
7.21.4.4 gga_mutex_	135
7.22 PowerManager Class Reference	135
7.22.1 Detailed Description	136
7.22.2 Constructor & Destructor Documentation	136
7.22.2.1 PowerManager()	136
7.22.3 Member Function Documentation	137
7.22.3.1 initialize()	137
7.22.3.2 read_device_ids()	137
7.22.3.3 get_current_charge_solar()	137
7.22.3.4 get_current_charge_usb()	137
7.22.3.5 get_current_charge_total()	137
7.22.3.6 get_current_draw()	138
7.22.3.7 get_voltage_battery()	138
7.22.3.8 get_voltage_5v()	138
7.22.3.9 configure()	138
7.22.3.10 is_charging_solar()	138
7.22.3.11 is_charging_usb()	139
7.22.3.12 check_power_alerts()	139
7.22.4 Member Data Documentation	140
7.22.4.1 SOLAR_CURRENT_THRESHOLD	140
7.22.4.2 USB_CURRENT_THRESHOLD	140
7.22.4.3 VOLTAGE_LOW_THRESHOLD	140
7.22.4.4 VOLTAGE_OVERCHARGE_THRESHOLD	140
7.22.4.5 FALL_RATE_THRESHOLD	140
7.22.4.6 FALLING_TREND_REQUIRED	140
7.22.4.7 ina3221_	140
7.22.4.8 initialized_	141
7.22.4.9 powerman_mutex_	141
7.22.4.10 charging_solar_active_	141
7.22.4.11 charging_usb_active_	141
7.23 SensorWrapper Class Reference	141
7.23.1 Detailed Description	142
7.23.2 Constructor & Destructor Documentation	142
7.23.2.1 SensorWrapper()	142
7.23.3 Member Function Documentation	143
7.23.3.1 get_instance()	143
7.23.3.2 sensor_init()	143
7.23.3.3 sensor_configure()	143

7.23.3.4 <code>sensor_read_data()</code>	144
7.23.4 Member Data Documentation	144
7.23.4.1 <code>sensors</code>	144
8 File Documentation	145
8.1 <code>build_number.h</code> File Reference	145
8.1.1 Macro Definition Documentation	145
8.1.1.1 <code>BUILD_NUMBER</code>	145
8.2 <code>build_number.h</code>	145
8.3 <code>includes.h</code> File Reference	146
8.4 <code>includes.h</code>	147
8.5 <code>lib/clock/DS3231.cpp</code> File Reference	147
8.6 <code>DS3231.cpp</code>	147
8.7 <code>lib/clock/DS3231.h</code> File Reference	151
8.7.1 Macro Definition Documentation	152
8.7.1.1 <code>DS3231_DEVICE_ADDRESS</code>	152
8.7.1.2 <code>DS3231_SECONDS_REG</code>	152
8.7.1.3 <code>DS3231_MINUTES_REG</code>	152
8.7.1.4 <code>DS3231_HOURS_REG</code>	152
8.7.1.5 <code>DS3231_DAY_REG</code>	152
8.7.1.6 <code>DS3231_DATE_REG</code>	152
8.7.1.7 <code>DS3231_MONTH_REG</code>	153
8.7.1.8 <code>DS3231_YEAR_REG</code>	153
8.7.1.9 <code>DS3231_CONTROL_REG</code>	153
8.7.1.10 <code>DS3231_CONTROL_STATUS_REG</code>	153
8.7.1.11 <code>DS3231_TEMPERATURE_MSB_REG</code>	153
8.7.1.12 <code>DS3231_TEMPERATURE_LSB_REG</code>	153
8.7.2 Enumeration Type Documentation	153
8.7.2.1 <code>days_of_week</code>	153
8.8 <code>DS3231.h</code>	154
8.9 <code>lib/comms/commands/clock_commands.cpp</code> File Reference	154
8.9.1 Macro Definition Documentation	155
8.9.1.1 <code>CLOCK_GROUP</code>	155
8.9.1.2 <code>TIME</code>	155
8.9.1.3 <code>TIMEZONE_OFFSET</code>	156
8.9.1.4 <code>CLOCK_SYNC_INTERVAL</code>	156
8.9.1.5 <code>LAST_SYNC_TIME</code>	156
8.10 <code>clock_commands.cpp</code>	156
8.11 <code>lib/comms/commands/commands.cpp</code> File Reference	158
8.12 <code>commands.cpp</code>	159
8.13 <code>lib/comms/commands/commands.h</code> File Reference	160
8.14 <code>commands.h</code>	162

8.15 lib/comms/commands/diagnostic_commands.cpp File Reference	162
8.16 diagnostic_commands.cpp	163
8.17 lib/comms/commands/event_commands.cpp File Reference	164
8.18 event_commands.cpp	165
8.19 lib/comms/commands/gps_commands.cpp File Reference	166
8.20 gps_commands.cpp	166
8.21 lib/comms/commands/power_commands.cpp File Reference	169
8.22 power_commands.cpp	169
8.23 lib/comms/commands/storage_commands.cpp File Reference	171
8.23.1 Macro Definition Documentation	172
8.23.1.1 MAX_BLOCK_SIZE	172
8.23.1.2 STORAGE_GROUP	172
8.23.1.3 START_COMMAND	172
8.23.1.4 DATA_COMMAND	172
8.23.1.5 END_COMMAND	172
8.23.1.6 LIST_FILES_COMMAND	172
8.23.1.7 MOUNT_COMMAND	173
8.24 storage_commands.cpp	173
8.25 lib/comms/commands/storage_commands_utils.cpp File Reference	175
8.25.1 Function Documentation	176
8.25.1.1 calculate_checksum()	176
8.25.1.2 send_data_block()	176
8.25.1.3 receive_ack()	176
8.26 storage_commands_utils.cpp	177
8.27 lib/comms/commands/storage_commands_utils.h File Reference	177
8.27.1 Function Documentation	178
8.27.1.1 calculate_checksum()	178
8.27.1.2 send_data_block()	178
8.27.1.3 receive_ack()	178
8.28 storage_commands_utils.h	179
8.29 lib/comms/communication.cpp File Reference	179
8.29.1 Function Documentation	179
8.29.1.1 initialize_radio()	179
8.29.2 Variable Documentation	180
8.29.2.1 outgoing	180
8.29.2.2 msgCount	180
8.29.2.3 lastSendTime	181
8.29.2.4 lastReceiveTime	181
8.29.2.5 lastPrintTime	181
8.29.2.6 interval	181
8.30 communication.cpp	181
8.31 lib/comms/communication.h File Reference	182

8.31.1 Function Documentation	183
8.31.1.1 initialize_radio()	183
8.31.1.2 on_receive()	183
8.31.1.3 handle_uart_input()	184
8.31.1.4 send_message()	185
8.31.1.5 send_frame()	185
8.31.1.6 send_frame_uart()	186
8.31.1.7 send_frame_lora()	186
8.31.1.8 split_and_send_message()	187
8.31.1.9 determine_unit()	187
8.32 communication.h	187
8.33 lib/comms/frame.cpp File Reference	188
8.33.1 Detailed Description	189
8.33.2 Typedef Documentation	189
8.33.2.1 CommandHandler	189
8.33.3 Variable Documentation	189
8.33.3.1 eventRegister	189
8.34 frame.cpp	189
8.35 lib/comms/protocol.h File Reference	191
8.35.1 Enumeration Type Documentation	192
8.35.1.1 OperationType	192
8.35.1.2 CommandAccessLevel	193
8.35.1.3 ValueUnit	193
8.35.1.4 ExceptionType	193
8.35.1.5 Interface	193
8.35.2 Function Documentation	194
8.35.2.1 exception_type_to_string()	194
8.35.2.2 operation_type_to_string()	194
8.35.2.3 string_to_operation_type()	195
8.35.2.4 hex_string_to_bytes()	195
8.35.2.5 value_unit_type_to_string()	196
8.35.3 Variable Documentation	197
8.35.3.1 FRAME_BEGIN	197
8.35.3.2 FRAME_END	198
8.35.3.3 DELIMITER	198
8.36 protocol.h	198
8.37 lib/comms/receive.cpp File Reference	199
8.37.1 Detailed Description	199
8.37.2 Function Documentation	200
8.37.2.1 on_receive()	200
8.37.2.2 handle_uart_input()	200
8.38 receive.cpp	201

8.39 lib/comms/send.cpp File Reference	202
8.39.1 Detailed Description	203
8.39.2 Function Documentation	203
8.39.2.1 send_message()	203
8.39.2.2 send_frame_lora()	204
8.39.2.3 send_frame_uart()	204
8.39.2.4 send_frame()	205
8.39.2.5 split_and_send_message()	205
8.40 send.cpp	206
8.41 lib/comms/utils_converters.cpp File Reference	207
8.41.1 Detailed Description	207
8.41.2 Function Documentation	207
8.41.2.1 exception_type_to_string()	207
8.41.2.2 value_unit_type_to_string()	208
8.41.2.3 operation_type_to_string()	209
8.41.2.4 string_to_operation_type()	210
8.41.2.5 hex_string_to_bytes()	211
8.42 utils_converters.cpp	211
8.43 lib/eventman/event_manager.cpp File Reference	212
8.43.1 Detailed Description	213
8.44 event_manager.cpp	213
8.45 lib/eventman/event_manager.h File Reference	215
8.45.1 Detailed Description	217
8.45.2 Macro Definition Documentation	217
8.45.2.1 EVENT_BUFFER_SIZE	217
8.45.2.2 EVENT_LOG_FILE	217
8.45.3 Function Documentation	217
8.45.3.1 to_string()	217
8.45.4 Variable Documentation	218
8.45.4.1 id	218
8.45.4.2 timestamp	218
8.45.4.3 group	218
8.45.4.4 event	218
8.46 event_manager.h	219
8.47 lib/location/gps_collector.cpp File Reference	221
8.47.1 Macro Definition Documentation	221
8.47.1.1 MAX_RAW_DATA_LENGTH	221
8.47.2 Function Documentation	222
8.47.2.1 splitString()	222
8.47.2.2 collect_gps_data()	222
8.47.3 Variable Documentation	223
8.47.3.1 nmea_data	223

8.48 gps_collector.cpp	223
8.49 lib/location/gps_collector.h File Reference	224
8.49.1 Function Documentation	224
8.49.1.1 collect_gps_data()	224
8.50 gps_collector.h	225
8.51 lib/location/NMEA/NMEA_data.cpp File Reference	226
8.51.1 Variable Documentation	226
8.51.1.1 nmea_data	226
8.52 NMEA_data.cpp	226
8.53 lib/location/NMEA/NMEA_data.h File Reference	227
8.53.1 Variable Documentation	228
8.53.1.1 nmea_data	228
8.54 NMEA_data.h	228
8.55 lib/pin_config.cpp File Reference	229
8.55.1 Variable Documentation	229
8.55.1.1 lora_cs_pin	229
8.55.1.2 lora_reset_pin	230
8.55.1.3 lora_irq_pin	230
8.55.1.4 lora_address_local	230
8.55.1.5 lora_address_remote	230
8.56 pin_config.cpp	230
8.57 lib/pin_config.h File Reference	231
8.57.1 Macro Definition Documentation	232
8.57.1.1 DEBUG_UART_PORT	232
8.57.1.2 DEBUG_UART_BAUD_RATE	232
8.57.1.3 DEBUG_UART_TX_PIN	232
8.57.1.4 DEBUG_UART_RX_PIN	232
8.57.1.5 MAIN_I2C_PORT	232
8.57.1.6 MAIN_I2C_SDA_PIN	233
8.57.1.7 MAIN_I2C_SCL_PIN	233
8.57.1.8 GPS_UART_PORT	233
8.57.1.9 GPS_UART_BAUD_RATE	233
8.57.1.10 GPS_UART_TX_PIN	233
8.57.1.11 GPS_UART_RX_PIN	233
8.57.1.12 GPS_POWER_ENABLE_PIN	233
8.57.1.13 BUFFER_SIZE	233
8.57.1.14 SD_SPI_PORT	234
8.57.1.15 SD_MISO_PIN	234
8.57.1.16 SD_MOSI_PIN	234
8.57.1.17 SD_SCK_PIN	234
8.57.1.18 SD_CS_PIN	234
8.57.1.19 SD_CARD_DETECT_PIN	234

8.57.1.20 SX1278_MISO	234
8.57.1.21 SX1278_CS	234
8.57.1.22 SX1278_SCK	235
8.57.1.23 SX1278_MOSI	235
8.57.1.24 SPI_PORT	235
8.57.1.25 READ_BIT	235
8.57.1.26 LORA_DEFAULT_SPI	235
8.57.1.27 LORA_DEFAULT_SPI_FREQUENCY	235
8.57.1.28 LORA_DEFAULT_SS_PIN	235
8.57.1.29 LORA_DEFAULT_RESET_PIN	235
8.57.1.30 LORA_DEFAULT_DIO0_PIN	236
8.57.1.31 PA_OUTPUT_RFO_PIN	236
8.57.1.32 PA_OUTPUT_PA_BOOST_PIN	236
8.57.2 Variable Documentation	236
8.57.2.1 lora_cs_pin	236
8.57.2.2 lora_reset_pin	236
8.57.2.3 lora_irq_pin	236
8.57.2.4 lora_address_local	236
8.57.2.5 lora_address_remote	237
8.58 pin_config.h	237
8.59 lib/powerman/INA3221/INA3221.cpp File Reference	238
8.59.1 Detailed Description	238
8.60 INA3221.cpp	238
8.61 lib/powerman/INA3221/INA3221.h File Reference	243
8.61.1 Detailed Description	244
8.61.2 Enumeration Type Documentation	244
8.61.2.1 ina3221_addr_t	244
8.61.2.2 ina3221_ch_t	245
8.61.2.3 ina3221_reg_t	245
8.61.2.4 ina3221_conv_time_t	246
8.61.2.5 ina3221_avg_mode_t	246
8.61.3 Variable Documentation	246
8.61.3.1 INA3221_CH_NUM	246
8.61.3.2 SHUNT_VOLTAGE_LSB_UV	247
8.62 INA3221.h	247
8.63 lib/powerman/PowerManager.cpp File Reference	249
8.64 PowerManager.cpp	249
8.65 lib/powerman/PowerManager.h File Reference	251
8.66 PowerManager.h	252
8.67 lib/sensors/BH1750/BH1750.cpp File Reference	253
8.68 BH1750.cpp	253
8.69 lib/sensors/BH1750/BH1750.h File Reference	254

8.69.1 Macro Definition Documentation	255
8.69.1.1 _BH1750_DEVICE_ID	255
8.69.1.2 _BH1750_MTREG_MIN	255
8.69.1.3 _BH1750_MTREG_MAX	256
8.69.1.4 _BH1750_DEFAULT_MTREG	256
8.70 BH1750.h	256
8.71 lib/sensors/BH1750/BH1750_WRAPPER.cpp File Reference	256
8.72 BH1750_WRAPPER.cpp	257
8.73 lib/sensors/BH1750/BH1750_WRAPPER.h File Reference	258
8.74 BH1750_WRAPPER.h	259
8.75 lib/sensors/BME280/BME280.cpp File Reference	259
8.76 BME280.cpp	260
8.77 lib/sensors/BME280/BME280.h File Reference	263
8.78 BME280.h	264
8.79 lib/sensors/BME280/BME280_WRAPPER.cpp File Reference	266
8.80 BME280_WRAPPER.cpp	266
8.81 lib/sensors/BME280/BME280_WRAPPER.h File Reference	267
8.82 BME280_WRAPPER.h	267
8.83 lib/sensors/HMC5883L/HMC5883L.cpp File Reference	268
8.84 HMC5883L.cpp	268
8.85 lib/sensors/HMC5883L/HMC5883L.h File Reference	269
8.86 HMC5883L.h	270
8.87 lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp File Reference	270
8.88 HMC5883L_WRAPPER.cpp	271
8.89 lib/sensors/HMC5883L/HMC5883L_WRAPPER.h File Reference	272
8.90 HMC5883L_WRAPPER.h	273
8.91 lib/sensors/ISensor.cpp File Reference	273
8.91.1 Detailed Description	274
8.92 ISensor.cpp	274
8.93 lib/sensors/ISensor.h File Reference	274
8.93.1 Enumeration Type Documentation	275
8.93.1.1 SensorType	275
8.93.1.2 SensorDataTypelIdentifier	276
8.94 ISensor.h	276
8.95 lib/sensors/MPU6050/MPU6050.cpp File Reference	277
8.96 MPU6050.cpp	277
8.97 lib/sensors/MPU6050/MPU6050.h File Reference	278
8.98 MPU6050.h	278
8.99 lib/sensors/MPU6050/MPU6050_WRAPPER.cpp File Reference	278
8.100 MPU6050_WRAPPER.cpp	278
8.101 lib/sensors/MPU6050/MPU6050_WRAPPER.h File Reference	278
8.102 MPU6050_WRAPPER.h	279

8.103 lib/storage/storage.cpp File Reference	280
8.103.1 Detailed Description	280
8.103.2 Function Documentation	281
8.103.2.1 fs_init()	281
8.103.3 Variable Documentation	282
8.103.3.1 sd_card_mounted	282
8.104 storage.cpp	282
8.105 lib/storage/storage.h File Reference	282
8.105.1 Function Documentation	283
8.105.1.1 fs_init()	283
8.105.1.2 fs_open_file()	284
8.105.1.3 fs_write_file()	284
8.105.1.4 fs_read_file()	285
8.105.1.5 fs_close_file()	285
8.105.1.6 fs_file_exists()	285
8.105.2 Variable Documentation	285
8.105.2.1 sd_card_mounted	285
8.106 storage.h	285
8.107 lib/utils.cpp File Reference	286
8.107.1 Detailed Description	287
8.107.2 Function Documentation	287
8.107.2.1 get_level_color()	287
8.107.2.2 get_level_prefix()	288
8.107.2.3 uart_print()	289
8.107.2.4 crc16()	290
8.107.3 Variable Documentation	291
8.107.3.1 uart_mutex	291
8.107.3.2 g_uart_verbosity	291
8.108 utils.cpp	291
8.109 lib/utils.h File Reference	292
8.109.1 Detailed Description	294
8.109.2 Macro Definition Documentation	294
8.109.2.1 ANSI_RED	294
8.109.2.2 ANSI_GREEN	294
8.109.2.3 ANSI_YELLOW	294
8.109.2.4 ANSI_BLUE	294
8.109.2.5 ANSI_CYAN	294
8.109.2.6 ANSI_RESET	294
8.109.3 Enumeration Type Documentation	294
8.109.3.1 VerbosityLevel	294
8.109.4 Function Documentation	295
8.109.4.1 uart_print()	295

8.109.4.2 crc16()	296
8.109.5 Variable Documentation	297
8.109.5.1 g_uart_verbosity	297
8.110 utils.h	298
8.111 main.cpp File Reference	298
8.111.1 Macro Definition Documentation	299
8.111.1.1 LOG_FILENAME	299
8.111.2 Function Documentation	299
8.111.2.1 process_pending_actions()	299
8.111.2.2 core1_entry()	300
8.111.2.3 init_systems()	300
8.111.2.4 main()	301
8.111.3 Variable Documentation	301
8.111.3.1 powerManager	301
8.111.3.2 systemClock	302
8.111.3.3 buffer	302
8.111.3.4 bufferIndex	302
8.112 main.cpp	302
8.113 test/comms/test_converters.cpp File Reference	304
8.113.1 Function Documentation	304
8.113.1.1 test_operation_type_conversion()	304
8.113.1.2 test_value_unit_type_conversion()	305
8.113.1.3 test_exception_type_conversion()	306
8.113.1.4 test_hex_string_conversion()	306
8.114 test_converters.cpp	307
8.115 test/comms/test_frame_build.cpp File Reference	307
8.115.1 Function Documentation	308
8.115.1.1 test_frame_build_success()	308
8.115.1.2 test_frame_build_error()	308
8.115.1.3 test_frame_build_info()	309
8.116 test_frame_build.cpp	309
8.117 test/comms/test_frame_coding.cpp File Reference	310
8.117.1 Function Documentation	310
8.117.1.1 test_frame_encode_basic()	310
8.117.1.2 test_frame_decode_basic()	311
8.117.1.3 test_frame_decode_invalid_header()	311
8.118 test_frame_coding.cpp	312
8.119 test/comms/test_frame_common.h File Reference	313
8.119.1 Function Documentation	313
8.119.1.1 create_test_frame()	313
8.120 test_frame_common.h	314
8.121 test/test_runner.cpp File Reference	314

8.121.1 Function Documentation	315
8.121.1.1 test_frame_encode_basic()	315
8.121.1.2 test_frame_decode_basic()	315
8.121.1.3 test_frame_decode_invalid_header()	316
8.121.1.4 test_frame_build_success()	316
8.121.1.5 test_frame_build_error()	317
8.121.1.6 test_frame_build_info()	318
8.121.1.7 test_operation_type_conversion()	318
8.121.1.8 test_value_unit_type_conversion()	319
8.121.1.9 test_exception_type_conversion()	320
8.121.1.10 test_hex_string_conversion()	320
8.121.1.11 main()	321
8.122 test_runner.cpp	321
Index	323

Chapter 1

Clock Commands

Member `handle_clock_sync_interval (const std::string ¶m, OperationType operationType)`

Command ID: 3.3

Member `handle_enable_gps_uart_passthrough (const std::string ¶m, OperationType operationType)`

Command ID: 7.2

Member `handle_enter_bootloader_mode (const std::string ¶m, OperationType operationType)`

Command ID: 2

Member `handle_file_download (const std::string ¶m, OperationType operationType)`

Command ID: 6.1

Member `handle_get_build_version (const std::string ¶m, OperationType operationType)`

Command ID: 1

Member `handle_get_commands_list (const std::string ¶m, OperationType operationType)`

Command ID: 0

Member `handle_get_current_charge_solar (const std::string ¶m, OperationType operationType)`

Command ID: 2.5

Member `handle_get_current_charge_total (const std::string ¶m, OperationType operationType)`

Command ID: 2.6

Member `handle_get_current_charge_usb (const std::string ¶m, OperationType operationType)`

Command ID: 2.4

Member `handle_get_current_draw (const std::string ¶m, OperationType operationType)`

Command ID: 2.7

Member `handle_get_event_count (const std::string ¶m, OperationType operationType)`

Command ID: 5.2

Member `handle_get_gga_data (const std::string ¶m, OperationType operationType)`

Command ID: 7.4

Member `handle_get_last_events (const std::string ¶m, OperationType operationType)`

Command ID: 5.1

Member `handle_get_last_sync_time (const std::string ¶m, OperationType operationType)`

Command ID: 3.7

Member `handle_get_power_manager_ids (const std::string ¶m, OperationType operationType)`

Command ID: 2.0

Member `handle_get_rmc_data (const std::string ¶m, OperationType operationType)`

Command ID: 7.3

Member `handle_get_voltage_5v (const std::string ¶m, OperationType operationType)`

Command ID: 2.3

Member `handle_get_voltage_battery (const std::string ¶m, OperationType operationType)`

Command ID: 2.2

Member `handle_gps_power_status (const std::string ¶m, OperationType operationType)`

Command ID: 7.1

Member `handle_list_files (const std::string ¶m, OperationType operationType)`

Command ID: 6.0

Member `handle_mount (const std::string ¶m, OperationType operationType)`

Command ID: 6.4

Member `handle_time (const std::string ¶m, OperationType operationType)`

Command ID: 3.0

Member `handle_timezone_offset (const std::string ¶m, OperationType operationType)`

Command ID: 3.1

Member `handle_verbosity (const std::string ¶m, 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	14
Diagnostic Commands	17
Event Commands	21
GPS Commands	23
Power Commands	27
Storage Commands	33
Frame Handling	36
Event Manager	41
INA3221 Power Monitor	50
Configuration Functions	50
Measurement Functions	60
Alert Functions	62

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

BH1750	69
BME280	75
BME280CalibParam	84
INA3221::conf_reg_t	90
DS3231	92
ds3231_data_t	99
EventEmitter	101
EventLog	102
EventManager	104
EventManagerImpl	109
FileHandle	112
Frame	112
HMC5883L	115
INA3221	120
ISensor	126
BH1750Wrapper	72
BME280Wrapper	87
HMC5883LWrapper	118
MPU6050Wrapper	131
INA3221::masken_reg_t	128
NMEAData	133
PowerManager	135
SensorWrapper	141

Chapter 4

Class Index

4.1 Class List

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

BH1750	69
BH1750Wrapper	72
BME280	75
BME280CalibParam	84
BME280Wrapper	87
INA3221::conf_reg_t Configuration register bit fields	90
DS3231	92
ds3231_data_t	99
EventEmitter Provides a static method for emitting events	101
EventLog Represents a single event log entry	102
EventManager Manages the event logging system	104
EventManagerImpl Implementation of the EventManager class	109
FileHandle	112
Frame Represents a communication frame used for data exchange	112
HMC5883L	115
HMC5883LWrapper	118
INA3221 INA3221 Triple-Channel Power Monitor driver class	120
ISensor	126
INA3221::masken_reg_t Mask/Enable register bit fields	128
MPU6050Wrapper	131
NMEAData	133
PowerManager	135
SensorWrapper Manages different sensor types and provides a unified interface for accessing sensor data	141

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

<code>build_number.h</code>	145
<code>includes.h</code>	146
<code>main.cpp</code>	298
<code>lib/pin_config.cpp</code>	229
<code>lib/pin_config.h</code>	231
<code>lib/utils.cpp</code> Implementation of utility functions for the Kabisat firmware	286
<code>lib/utils.h</code> Utility functions and definitions for the Kabisat firmware	292
<code>lib/clock/DS3231.cpp</code>	147
<code>lib/clock/DS3231.h</code>	151
<code>lib/comms/communication.cpp</code>	179
<code>lib/comms/communication.h</code>	182
<code>lib/comms/frame.cpp</code> Implements functions for encoding, decoding, building, and processing Frames	188
<code>lib/comms/protocol.h</code>	191
<code>lib/comms/receive.cpp</code> Implements functions for receiving and processing data, including LoRa and UART input	199
<code>lib/comms/send.cpp</code> Implements functions for sending data, including LoRa messages and Frames	202
<code>lib/comms/utils_converters.cpp</code> Implements utility functions for converting between different data types	207
<code>lib/comms/commands/clock_commands.cpp</code>	154
<code>lib/comms/commands/commands.cpp</code>	158
<code>lib/comms/commands/commands.h</code>	160
<code>lib/comms/commands/diagnostic_commands.cpp</code>	162
<code>lib/comms/commands/event_commands.cpp</code>	164
<code>lib/comms/commands/gps_commands.cpp</code>	166
<code>lib/comms/commands/power_commands.cpp</code>	169
<code>lib/comms/commands/storage_commands.cpp</code>	171
<code>lib/comms/commands/storage_commands_utils.cpp</code>	175
<code>lib/comms/commands/storage_commands_utils.h</code>	177
<code>lib/eventman/event_manager.cpp</code> Implements the event management system for the Kabisat firmware	212
<code>lib/eventman/event_manager.h</code> Manages the event logging system for the Kabisat firmware	215

lib/location/gps_collector.cpp	221
lib/location/gps_collector.h	224
lib/location/NMEA/NMEA_data.cpp	226
lib/location/NMEA/NMEA_data.h	227
lib/powerman/PowerManager.cpp	249
lib/powerman/PowerManager.h	251
lib/powerman/INA3221/INA3221.cpp Implementation of the INA3221 power monitor driver	238
lib/powerman/INA3221/INA3221.h Header file for the INA3221 triple-channel power monitor driver	243
lib/sensors/ISensor.cpp Implements the SensorWrapper class for managing different sensor types	273
lib/sensors/ISensor.h	274
lib/sensors/BH1750/BH1750.cpp	253
lib/sensors/BH1750/BH1750.h	254
lib/sensors/BH1750/BH1750_WRAPPER.cpp	256
lib/sensors/BH1750/BH1750_WRAPPER.h	258
lib/sensors/BME280/BME280.cpp	259
lib/sensors/BME280/BME280.h	263
lib/sensors/BME280/BME280_WRAPPER.cpp	266
lib/sensors/BME280/BME280_WRAPPER.h	267
lib/sensors/HMC5883L/HMC5883L.cpp	268
lib/sensors/HMC5883L/HMC5883L.h	269
lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp	270
lib/sensors/HMC5883L/HMC5883L_WRAPPER.h	272
lib/sensors/MPU6050/MPU6050.cpp	277
lib/sensors/MPU6050/MPU6050.h	278
lib/sensors/MPU6050/MPU6050_WRAPPER.cpp	278
lib/sensors/MPU6050/MPU6050_WRAPPER.h	278
lib/storage/storage.cpp Implements file system operations for the Kubisat firmware	280
lib/storage/storage.h	282
test/test_runner.cpp	314
test/comms/test_converters.cpp	304
test/comms/test_frame_build.cpp	307
test/comms/test_frame_coding.cpp	310
test/comms/test_frame_common.h	313

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 ¶m, OperationType operationType)
Handler for getting and setting system time.
- std::vector< Frame > handle_timezone_offset (const std::string ¶m, OperationType operationType)
Handler for getting and setting timezone offset.
- std::vector< Frame > handle_clock_sync_interval (const std::string ¶m, OperationType operationType)
Handler for getting and setting clock synchronization interval.
- std::vector< Frame > handle_get_last_sync_time (const std::string ¶m, 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"

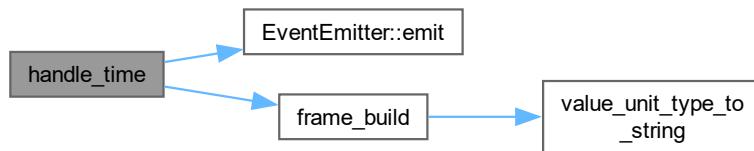
SET: **KBST;0;SET;3;0;TIMESTAMP;KBST**

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 89 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

<i>param</i>	For SET: Sync interval in seconds, for GET: empty string
<i>operationType</i>	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 138 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

<i>param</i>	Empty string expected
<i>operationType</i>	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 186 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 ¶m, OperationType operationType)`
Executes a command based on its key.

Variables

- `CommandMap commandHandlers`
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

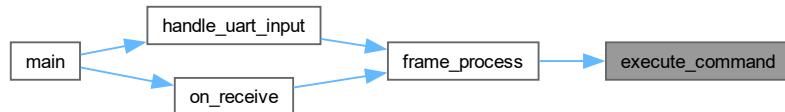
Looks up the command handler in commandHandlers map and executes it

Definition at line 63 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 commandHandlers

CommandMap commandHandlers

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>(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>(1)), handle_file_download},
{{(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 ¶m, [OperationType](#) operationType)
Handler for listing all available commands on UART.
- std::vector< [Frame](#) > [handle_get_build_version](#) (const std::string ¶m, [OperationType](#) operationType)
Get firmware build version.
- std::vector< [Frame](#) > [handle_verbosity](#) (const std::string ¶m, [OperationType](#) operationType)
Handles setting or getting the UART verbosity level.
- std::vector< [Frame](#) > [handle_enter_bootloader_mode](#) (const std::string ¶m, [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: 1Definition at line 61 of file [diagnostic_commands.cpp](#).

Here is the call 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.

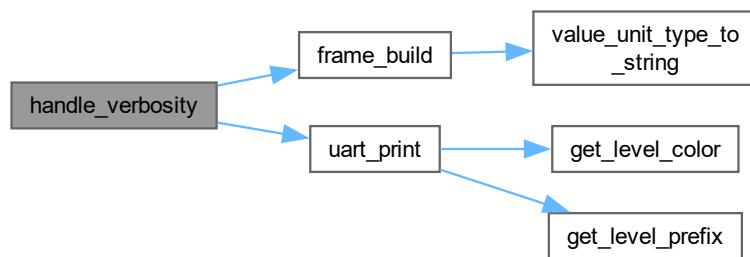
KBST;0;SET;1;8;[level];TSBK - Sets the verbosity level.

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

Command Command ID: 1.8

Definition at line 100 of file [diagnostic_commands.cpp](#).

Here is the call 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

<code>param</code>	Empty string expected
<code>operationType</code>	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: 2Definition at line 136 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 ¶m, OperationType operationType)`
Handler for retrieving last N events from the event log.
- `std::vector< Frame > handle_get_event_count (const std::string ¶m, 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)
<i>operationType</i>	GET

Returns

Frame containing:

- Success: Hex-encoded events in format IIIITTTTTTGGE where:
 - III: Event ID (16-bit)
 - TTTTTTTT: Unix Timestamp (32-bit)
 - GG: Event Group (8-bit)
 - EE: Event Type (8-bit)
- Error: "INVALID OPERATION", "INVALID COUNT", or "INVALID PARAMETER"

Note

KBST;0;GET;5;1;20;TSBK

Returns up to 10 most recent events

Command Command ID: 5.1

Definition at line 29 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 86 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 ¶m, **OperationType** operationType)
Handler for controlling GPS module power state.
- std::vector< **Frame** > **handle_enable_gps_uart_passthrough** (const std::string ¶m, **OperationType** operationType)
Handler for enabling GPS transparent mode (UART pass-through)
- std::vector< **Frame** > **handle_get_rmc_data** (const std::string ¶m, **OperationType** operationType)
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- std::vector< **Frame** > **handle_get_gga_data** (const std::string ¶m, **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

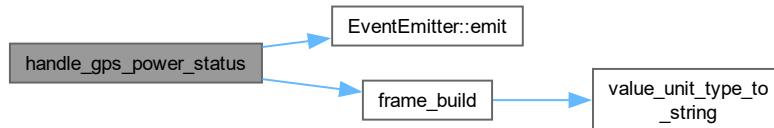
KBST;0;SET;7;1;POWER;TSBK

POWER - 0 - OFF, 1 - ON

Command Command ID: 7.1

Definition at line 26 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

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

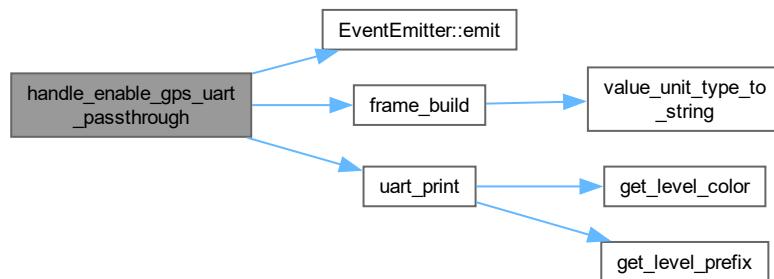
Enters a pass-through mode where UART communication is bridged directly to GPS

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

Command Command ID: 7.2

Definition at line 83 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 182 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 226 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 ¶m, OperationType operationType)
Handler for retrieving Power Manager IDs.
- std::vector< Frame > [handle_get_voltage_battery](#) (const std::string ¶m, OperationType operationType)
Handler for getting battery voltage.
- std::vector< Frame > [handle_get_voltage_5v](#) (const std::string ¶m, OperationType operationType)
Handler for getting 5V rail voltage.
- std::vector< Frame > [handle_get_current_charge_usb](#) (const std::string ¶m, OperationType operationType)
Handler for getting USB charge current.
- std::vector< Frame > [handle_get_current_charge_solar](#) (const std::string ¶m, OperationType operationType)
Handler for getting solar panel charge current.
- std::vector< Frame > [handle_get_current_charge_total](#) (const std::string ¶m, OperationType operationType)
Handler for getting total charge current.
- std::vector< Frame > [handle_get_current_draw](#) (const std::string ¶m, 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 21 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

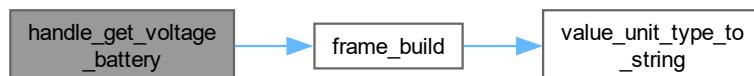
KBST;0;GET;2;2;;TSBK

This command is used to retrieve the battery voltage

Command Command ID: 2.2

Definition at line 51 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

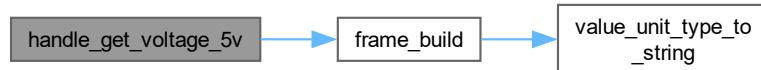
KBST;0;GET;2;3;;TSBK

This command is used to retrieve the 5V rail voltage

Command Command ID: 2.3

Definition at line 81 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

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

Returns

Vector of Frames containing:

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

Note

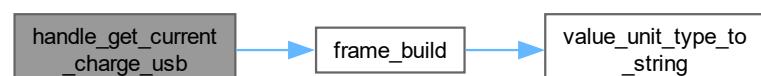
KBST;0;GET;2;4;;TSBK

This command is used to retrieve the USB charge current

Command Command ID: 2.4

Definition at line 111 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

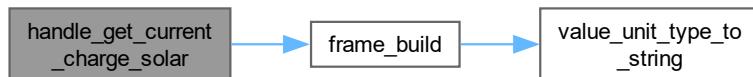
This command is used to retrieve the solar panel charge current

Command

Command ID: 2.5

Definition at line 141 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 171 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 201 of file [power_commands.cpp](#).

Here is the call graph for this function:



6.7 Storage Commands

Commands for interacting with the SD card storage.

Functions

- `std::vector< Frame > handle_file_download (const std::string ¶m, OperationType operationType)`
Handles the file download command.
- `std::vector< Frame > handle_list_files (const std::string ¶m, OperationType operationType)`
Handles the list files command.
- `std::vector< Frame > handle_mount (const std::string ¶m, OperationType operationType)`
Handles the SD card mount/unmount command.

6.7.1 Detailed Description

Commands for interacting with the SD card storage.

6.7.2 Function Documentation

6.7.2.1 handle_file_download()

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

Handles the file download command.

This function reads a file from the SD card and sends it to the ground station in blocks over LoRa. The ground station must acknowledge each block before the next block is sent. A checksum is calculated and sent at the end of the transmission to verify data integrity.

Parameters

<i>param</i>	The filename to download.
<i>operationType</i>	The operation type (must be GET).

Returns

A vector of Frames indicating the result of the operation.

- Success: Frame with "File download complete" message.
- Error: Frame with error message (e.g., "File not found", "ACK timeout").

Note

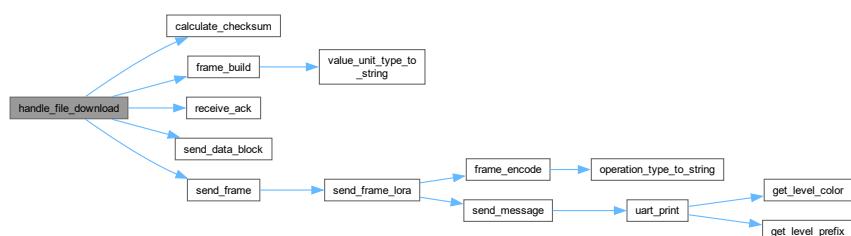
KBST;0;GET;6;1;[filename];TSBK

Example: **KBST;0;GET;6;1;test.txt;TSBK** - Downloads the file "test.txt".

Command Command ID: 6.1

Definition at line 43 of file [storage_commands.cpp](#).

Here is the call graph for this function:



6.7.2.2 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 124 of file [storage_commands.cpp](#).

Here is the call graph for this function:

**6.7.2.3 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

<i>param</i>	"0" to unmount, "1" to mount.
<i>operationType</i>	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

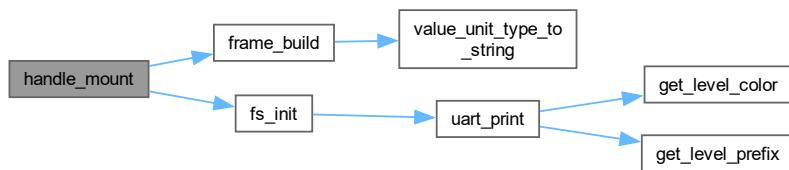
KBST;0;SET;6;4;[0|1];TSBK

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

Command Command ID: 6.4

Definition at line 206 of file [storage_commands.cpp](#).

Here is the call graph for this function:



6.8 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.

6.8.1 Detailed Description

Functions for encoding, decoding and building communication frames.

6.8.2 Function Documentation

6.8.2.1 `frame_encode()`

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

Encodes a `Frame` instance into a string.

Parameters

<code>frame</code>	The Frame 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;

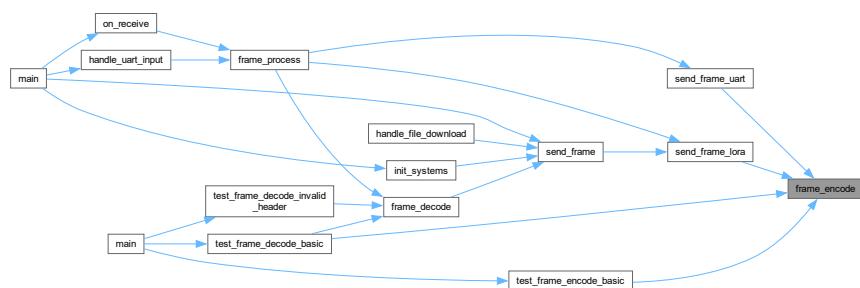
std::string encoded = frame_encode(myFrame);
// encoded will be "KBST;0;GET;1;1;;TSBK"
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

6.8.2.2 `frame_decode()`

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

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

Parameters

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

Returns

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

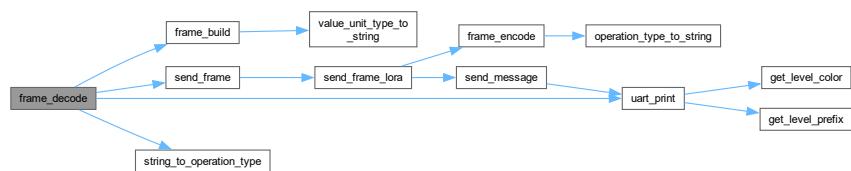
Exceptions

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

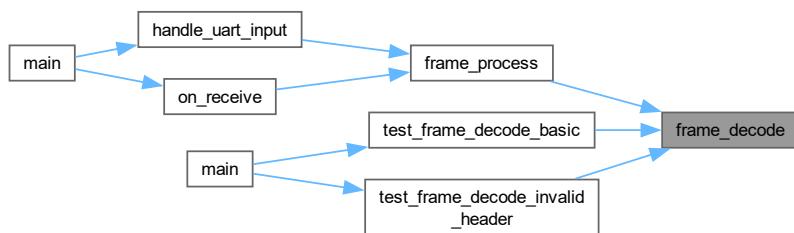
The decoded string is expected to be in the format: FRAME_BEGIN;direction;operationType;group;command;value;unit;FRAME_END

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.8.2.3 `frame_process()`

```

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

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

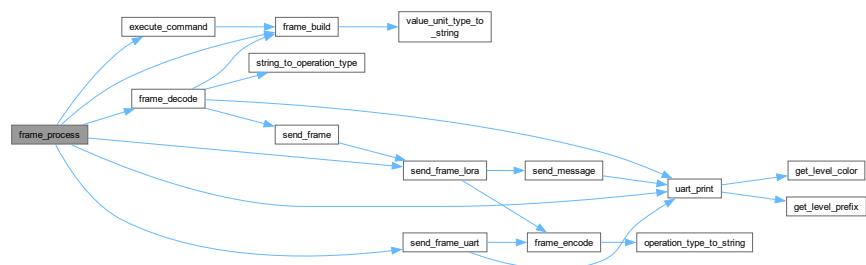
Parameters

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

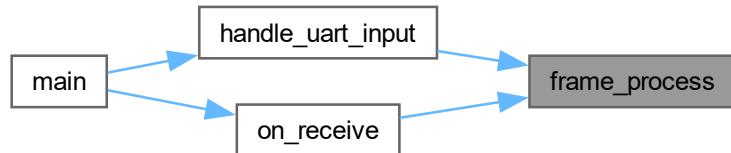
Decodes the frame data, extracts the command key, and executes the corresponding command. Sends the response frame. If an error occurs, an error frame is built and sent.

Definition at line 118 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

6.8.2.4 `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

<code>result</code>	The execution result.
<code>group</code>	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

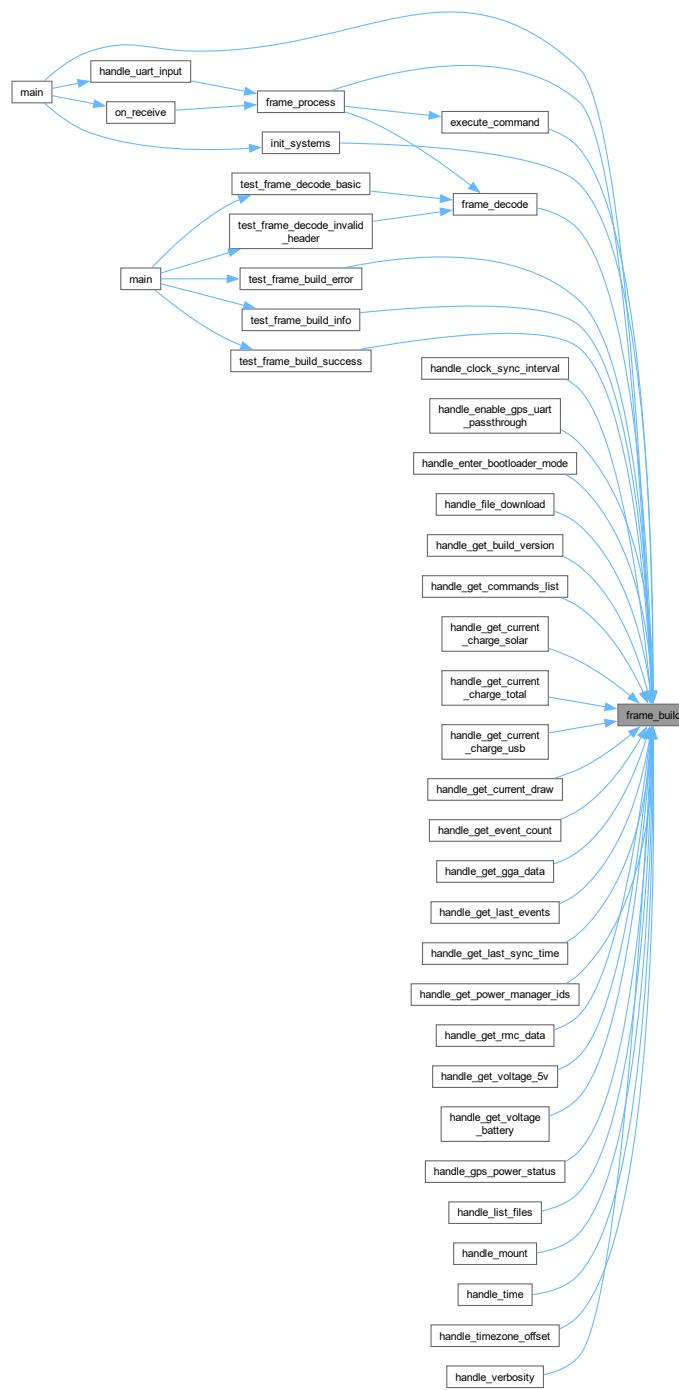
The [Frame](#) instance.

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.9 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 }
Represents specific clock-related events.

Functions

- void `check_power_events` (`PowerManager` &`pm`)
Checks power statuses and triggers events based on voltage trends.
- 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 eventLogId` = 0
Global event log ID counter.
- static `PowerEvent lastPowerState = PowerEvent::LOW_BATTERY`
Stores the last known power state.
- static constexpr float `FALL_RATE_THRESHOLD` = -0.02f
Threshold for detecting a falling voltage rate.
- static constexpr int `FALLING_TREND_REQUIRED` = 3
Number of consecutive falling voltage readings required to trigger a power falling event.
- static constexpr float `VOLTAGE_LOW_THRESHOLD` = 4.7f
Voltage threshold for detecting a low battery condition.
- static constexpr float `VOLTAGE_OVERCHARGE_THRESHOLD` = 5.3f
Voltage threshold for detecting an overcharge condition.
- static int `fallingTrendCount` = 0
Counter for consecutive falling voltage readings.
- bool `lastSolarState` = false
Stores the last known solar charging state.
- bool `lastUSBState` = false
Stores the last known USB connection state.
- `DS3231 systemClock`
External declaration of the system clock.
- `EventManagerImpl eventManager`
Global instance of the `EventManager` implementation.
- class `EventManager __attribute__ ((packed))`
- `EventManagerImpl eventManager`
Global instance of the `EventManagerImpl` class.

6.9.1 Detailed Description

Classes and functions for managing event logging.

6.9.2 Enumeration Type Documentation

6.9.2.1 EventGroup

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

Represents the group to which an event belongs.

Enumerator

SYSTEM	System events.
POWER	Power-related events.
COMMS	Communication-related events.
GPS	GPS-related events.
CLOCK	Clock-related events.

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

6.9.2.2 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 43 of file [event_manager.h](#).

6.9.2.3 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 60 of file [event_manager.h](#).

6.9.2.4 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 83 of file [event_manager.h](#).

6.9.2.5 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 100 of file [event_manager.h](#).

6.9.2.6 ClockEvent

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

Represents specific clock-related events.

Enumerator

CHANGED	Clock changed event.
GPS_SYNC	Clock synchronized with GPS event.

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

6.9.3 Function Documentation

6.9.3.1 check_power_events()

```
void check_power_events (
    PowerManager & pm)
```

Checks power statuses and triggers events based on voltage trends.

Parameters

<i>pm</i>	Reference to the PowerManager object.
-----------	---

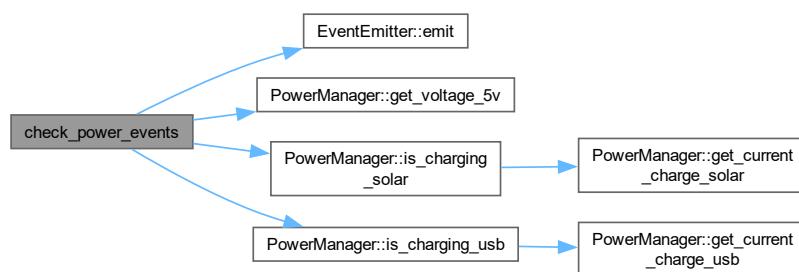
Monitors the 5V voltage level, detects falling voltage trends, and triggers events for low battery, overcharge, and normal power conditions. Also checks solar charging and USB connection states.

Parameters

<i>pm</i>	Reference to the PowerManager object.
-----------	---

Definition at line 154 of file [event_manager.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



6.9.3.2 [__attribute__\(\)](#)

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

6.9.3.3 [log_event\(\)](#)

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

Logs an event.

Logs an event to the event buffer.

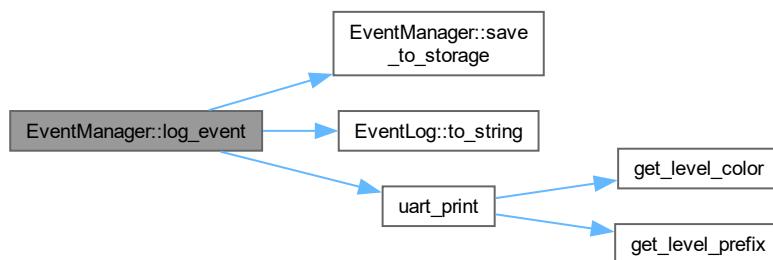
Parameters

<i>group</i>	The event group.
<i>event</i>	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 93 of file [event_manager.cpp](#).

Here is the call graph for this function:

**6.9.3.4 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 128 of file [event_manager.cpp](#).

6.9.4 Variable Documentation

6.9.4.1 eventLogId

```
volatile uint16_t eventLogId = 0
```

Global event log ID counter.

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

6.9.4.2 lastPowerState

```
PowerEvent lastPowerState = PowerEvent::LOW_BATTERY [static]
```

Stores the last known power state.

Definition at line [28](#) of file [event_manager.cpp](#).

6.9.4.3 FALL_RATE_THRESHOLD

```
float FALL_RATE_THRESHOLD = -0.02f [static], [constexpr]
```

Threshold for detecting a falling voltage rate.

Definition at line [34](#) of file [event_manager.cpp](#).

6.9.4.4 FALLING_TREND_REQUIRED

```
int FALLING_TREND_REQUIRED = 3 [static], [constexpr]
```

Number of consecutive falling voltage readings required to trigger a power falling event.

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

6.9.4.5 VOLTAGE_LOW_THRESHOLD

```
float VOLTAGE_LOW_THRESHOLD = 4.7f [static], [constexpr]
```

Voltage threshold for detecting a low battery condition.

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

6.9.4.6 VOLTAGE_OVERCHARGE_THRESHOLD

```
float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f [static], [constexpr]
```

Voltage threshold for detecting an overcharge condition.

Definition at line [52](#) of file [event_manager.cpp](#).

6.9.4.7 fallingTrendCount

```
int fallingTrendCount = 0 [static]
```

Counter for consecutive falling voltage readings.

Definition at line 58 of file [event_manager.cpp](#).

6.9.4.8 lastSolarState

```
bool lastSolarState = false
```

Stores the last known solar charging state.

Definition at line 64 of file [event_manager.cpp](#).

6.9.4.9 lastUSBState

```
bool lastUSBState = false
```

Stores the last known USB connection state.

Definition at line 70 of file [event_manager.cpp](#).

6.9.4.10 systemClock

```
DS3231 systemClock [extern]
```

External declaration of the system clock.

6.9.4.11 eventManager [1/2]

```
EventManagerImpl eventManager
```

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

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

Definition at line 82 of file [event_manager.cpp](#).

6.9.4.12 __attribute__

```
class EventManager __attribute__
```

6.9.4.13 eventManager [2/2]

`EventManagerImpl eventManager [extern]`

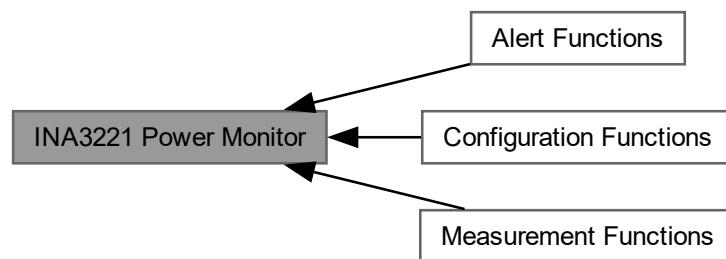
Global instance of the `EventManagerImpl` class.

Global instance of the `EventManagerImpl` class.

Definition at line 82 of file `event_manager.cpp`.

6.10 INA3221 Power Monitor

Collaboration diagram for INA3221 Power Monitor:



Topics

- Configuration Functions
- Measurement Functions
- Alert Functions

6.10.1 Detailed Description

6.10.2 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 ()`
Get the manufacturer ID of the device.
- `uint16_t INA3221::get_die_id ()`
Get the die 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.

6.10.2.1 Detailed Description

Functions for configuring the `INA3221` device

6.10.2.2 Function Documentation

6.10.2.2.1 `INA3221()`

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

Constructor for `INA3221` class.

Parameters

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

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

6.10.2.2.2 begin()

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

Initialize the [INA3221](#) device.

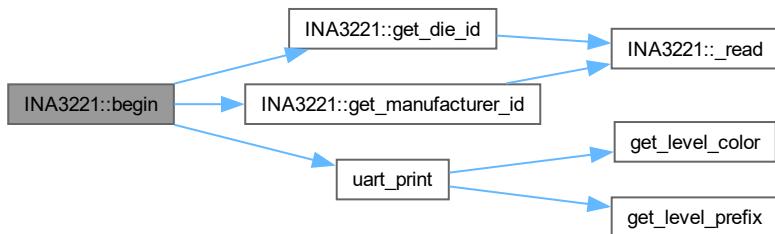
Returns

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:



6.10.2.2.3 reset()

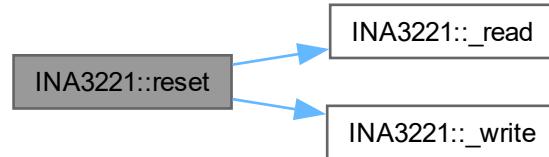
```
void INA3221::reset ()
```

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:



6.10.2.2.4 `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:



6.10.2.2.5 get_die_id()

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

Get the die ID of the device.

Returns

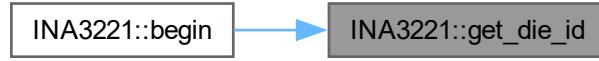
16-bit die ID (should be 0x3220)

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.2.2.6 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:

**6.10.2.2.7 set_mode_power_down()**

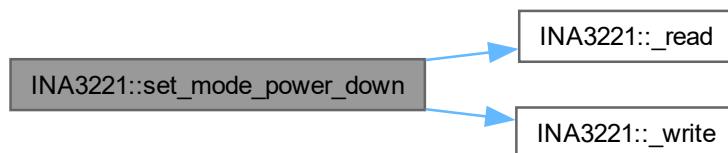
```
void INA3221::set_mode_power_down ()
```

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:

**6.10.2.2.8 set_mode_continuous()**

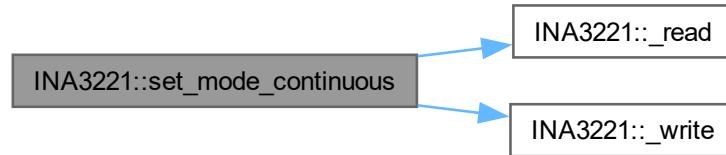
```
void INA3221::set_mode_continuous ()
```

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:



6.10.2.2.9 `set_mode_triggered()`

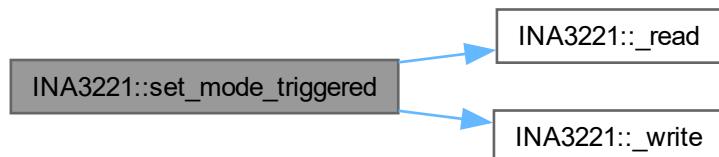
```
void INA3221::set_mode_triggered ()
```

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:



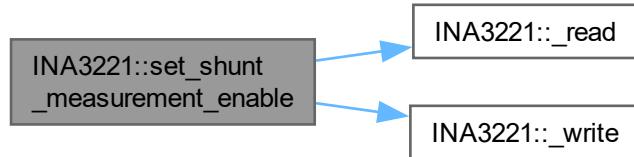
6.10.2.2.10 `set_shunt_measurement_enable()`

```
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:



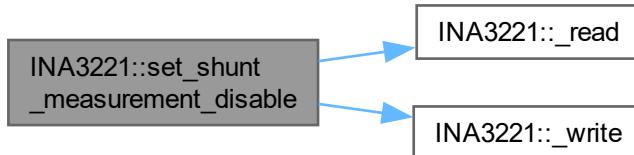
6.10.2.2.11 `set_shunt_measurement_disable()`

```
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:



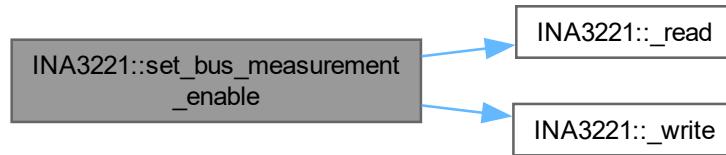
6.10.2.2.12 `set_bus_measurement_enable()`

```
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:



6.10.2.2.13 `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:



6.10.2.2.14 `set_averaging_mode()`

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

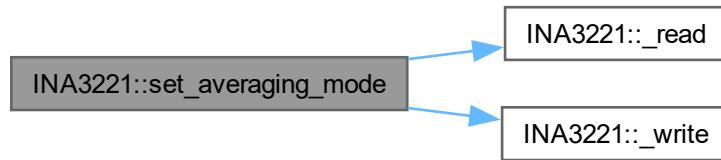
Set the averaging mode for measurements.

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:



6.10.2.2.15 set_bus_conversion_time()

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

Set bus voltage conversion time.

Parameters

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

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

Here is the call graph for this function:



6.10.2.2.16 set_shunt_conversion_time()

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

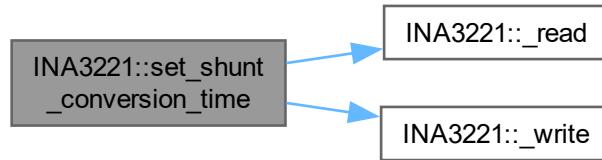
Set shunt voltage conversion time.

Parameters

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

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

Here is the call graph for this function:



6.10.3 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.

6.10.3.1 Detailed Description

Functions for reading voltage, current and power measurements

6.10.3.2 Function Documentation

6.10.3.2.1 `get_shunt_voltage()`

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

Get shunt voltage for a specific channel.

Parameters

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

Returns

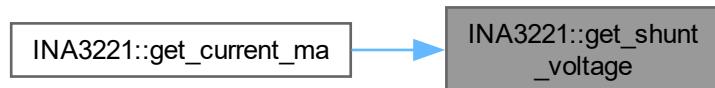
Shunt voltage in microvolts (μ V)

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.3.2.2 `get_current_ma()`

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

Get current for a specific channel.

Parameters

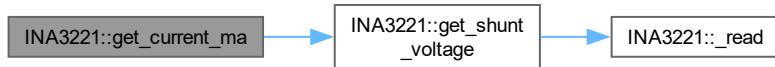
<i>channel</i>	Channel number (1-3)
----------------	----------------------

Returns

Current in millamps (mA)

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

Here is the call graph for this function:



6.10.3.2.3 get_voltage()

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

Get bus voltage for a specific channel.

Parameters

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

Returns

Voltage in volts (V)

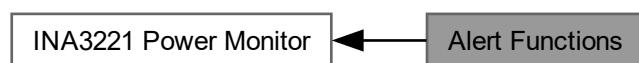
Definition at line 330 of file [INA3221.cpp](#).

Here is the call graph for this function:



6.10.4 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.

6.10.4.1 Detailed Description

Functions for configuring and reading alert conditions

6.10.4.2 Function Documentation

6.10.4.2.1 `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

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

Definition at line 360 of file `INA3221.cpp`.

Here is the call graph for this function:



6.10.4.2.2 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</i> ← _v	Voltage threshold in volts

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

Here is the call graph for this function:



6.10.4.2.3 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</i> ← _v	Upper voltage threshold in volts
<i>voltage_lower</i> ← _v	Lower voltage threshold in volts

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

Here is the call graph for this function:



6.10.4.2.4 enable_alerts()

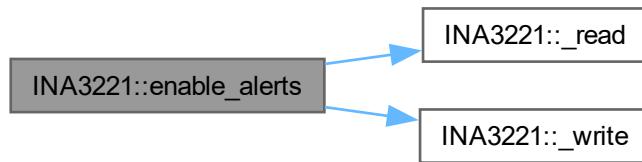
```
void INA3221::enable_alerts ()
```

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:



6.10.4.2.5 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:



6.10.4.2.6 `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:

**6.10.4.2.7 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:

**6.10.4.2.8 set_alert_latch()**

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

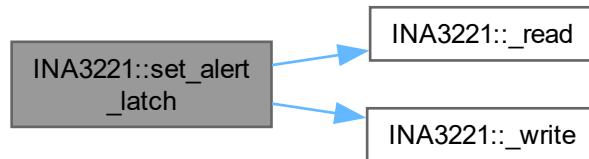
Set alert latch mode.

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:



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()

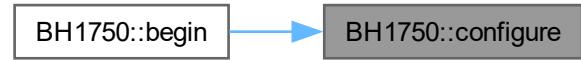
```
void BH1750::configure (
    Mode mode)
```

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](#).

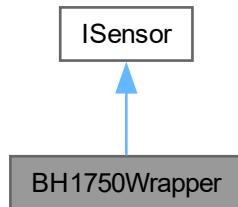
The documentation for this class was generated from the following files:

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

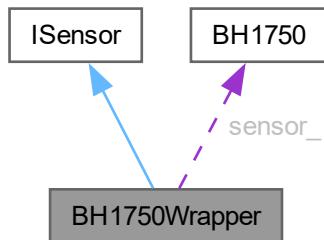
7.2 BH1750Wrapper Class Reference

```
#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)`

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](#).

7.2.4 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](#).

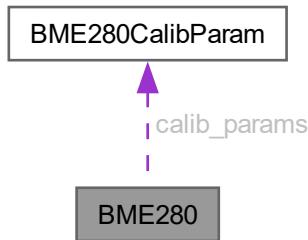
The documentation for this class was generated from the following files:

- 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](#).

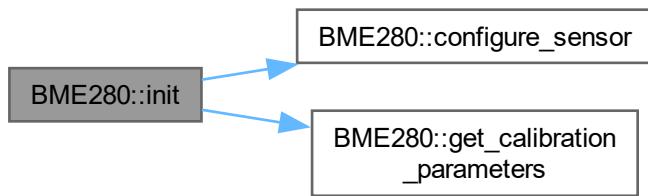
7.3.3 Member Function Documentation

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](#).

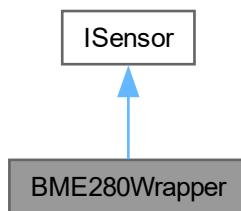
The documentation for this struct was generated from the following file:

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

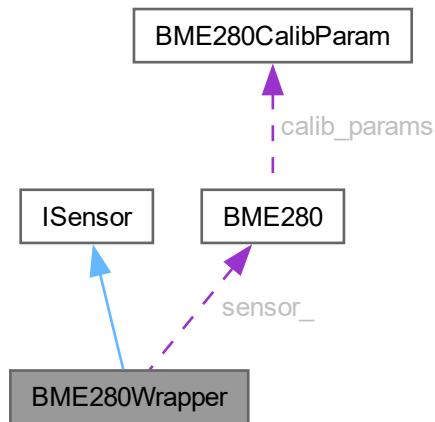
7.5 BME280Wrapper Class Reference

```
#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

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](#).

7.5.4 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

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

Public Member Functions

- [DS3231](#) (i2c_inst_t *i2c_instance)
- int [set_time](#) (ds3231_data_t *data)
- int [get_time](#) (ds3231_data_t *data)
- int [read_temperature](#) (float *resolution)
- int [set_unix_time](#) (time_t unix_time)
- time_t [get_unix_time](#) ()
- int [clock_enable](#) ()

Private Member Functions

- int [i2c_read_reg](#) (uint8_t reg_addr, size_t length, uint8_t *data)

Library function to read a specific I2C register address.
- int [i2c_write_reg](#) (uint8_t reg_addr, size_t length, uint8_t *data)

Library function to write to a specific I2C register address.
- uint8_t [bin_to_bcd](#) (const uint8_t data)

Library function that takes an 8-bit unsigned integer and converts it into a Binary Coded Decimal number that can be written to [DS3231](#) registers.
- uint8_t [bcd_to_bin](#) (const uint8_t bcd)

Library function that takes a BCD number and converts it to an unsigned 8-bit integer.

Private Attributes

- i2c_inst_t * [i2c](#)
- uint8_t [ds3231_addr](#)
- recursive_mutex_t [clock_mutex_](#)

7.7.1 Detailed Description

Definition at line 48 of file [DS3231.h](#).

7.7.2 Constructor & Destructor Documentation

7.7.2.1 DS3231()

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

Definition at line 6 of file [DS3231.cpp](#).

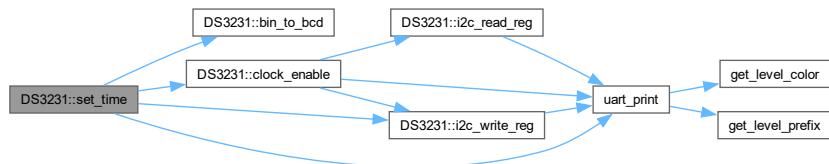
7.7.3 Member Function Documentation

7.7.3.1 set_time()

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

Definition at line 11 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

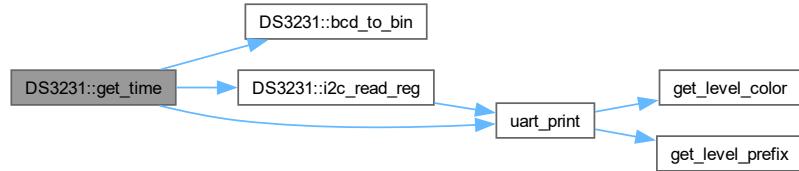


7.7.3.2 get_time()

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

Definition at line 68 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

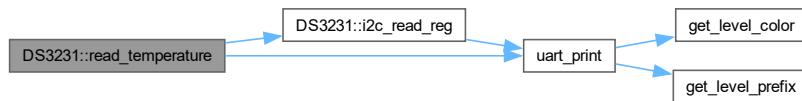


7.7.3.3 read_temperature()

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

Definition at line 110 of file [DS3231.cpp](#).

Here is the call graph for this function:

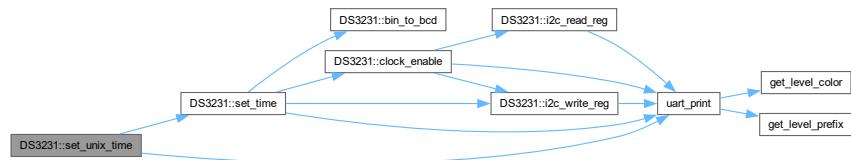


7.7.3.4 set_unix_time()

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

Definition at line 217 of file [DS3231.cpp](#).

Here is the call graph for this function:

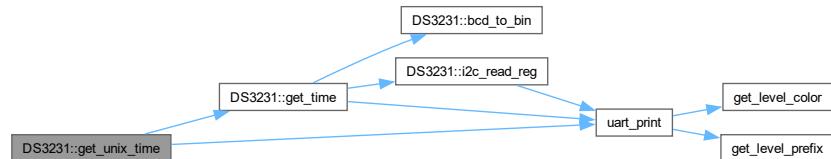


7.7.3.5 get_unix_time()

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

Definition at line 237 of file [DS3231.cpp](#).

Here is the call graph for this function:

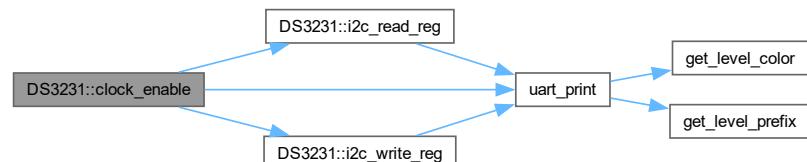


7.7.3.6 clock_enable()

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

Definition at line 265 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.7 i2c_read_reg()

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

Library function to read a specific I2C register address.

Parameters

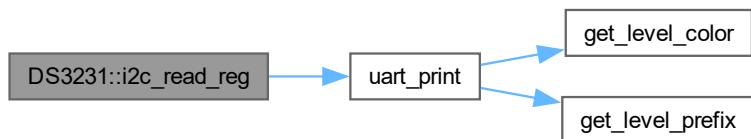
in	<i>reg_addr</i>	Register address to be read.
in	<i>length</i>	length of the data in bytes to be read.
out	<i>data</i>	Buffer to store the read data.

Returns

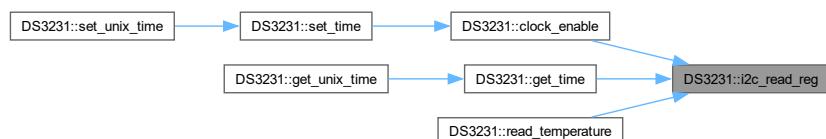
0 if successful, -1 if i2c failure.

Definition at line 136 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.8 i2c_write_reg()

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

Library function to write to a specific I2C register address.

Parameters

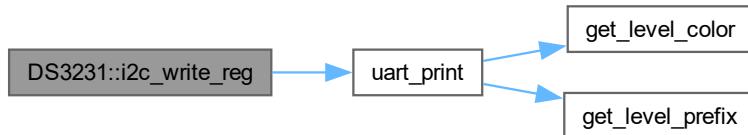
in	<i>reg_addr</i>	Register address to be written.
in	<i>length</i>	Length of the data to be written in bytes.
in	<i>data</i>	Pointer to the data buffer.

Returns

0 if successful, -1 if i2c failure.

Definition at line 171 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.9 bin_to_bcd()

```
uint8_t DS3231::bin_to_bcd (
    const uint8_t data) [private]
```

Library function that takes an 8-bit unsigned integer and converts it into a Binary Coded Decimal number that can be written to [DS3231](#) registers.

Parameters

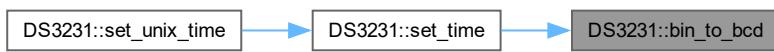
in	<i>data</i>	Number to be converted.
----	-------------	-------------------------

Returns

Number in BCD form.

Definition at line 199 of file [DS3231.cpp](#).

Here is the caller graph for this function:



7.7.3.10 bcd_to_bin()

```
uint8_t DS3231::bcd_to_bin (
    const uint8_t bcd) [private]
```

Library function that takes a BCD number and converts it to an unsigned 8-bit integer.

Parameters

in	<i>bcd</i>	BCD number to be converted.
----	------------	-----------------------------

Returns

Unsigned 8-bit integer.

Definition at line 211 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]
```

Definition at line 61 of file [DS3231.h](#).

7.7.4.2 ds3231_addr

```
uint8_t DS3231::ds3231_addr [private]
```

Definition at line 62 of file [DS3231.h](#).

7.7.4.3 clock_mutex_

```
recursive_mutex_t DS3231::clock_mutex_ [private]
```

Definition at line 70 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

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

Public Attributes

- `uint8_t seconds`
- `uint8_t minutes`
- `uint8_t hours`
- `uint8_t day`
- `uint8_t date`
- `uint8_t month`
- `uint8_t year`
- `bool century`

7.8.1 Detailed Description

Definition at line 37 of file [DS3231.h](#).

7.8.2 Member Data Documentation

7.8.2.1 seconds

```
uint8_t ds3231_data_t::seconds
```

Definition at line 38 of file [DS3231.h](#).

7.8.2.2 minutes

```
uint8_t ds3231_data_t::minutes
```

Definition at line 39 of file [DS3231.h](#).

7.8.2.3 hours

```
uint8_t ds3231_data_t::hours
```

Definition at line 40 of file [DS3231.h](#).

7.8.2.4 day

```
uint8_t ds3231_data_t::day
```

Definition at line 41 of file [DS3231.h](#).

7.8.2.5 date

```
uint8_t ds3231_data_t::date
```

Definition at line 42 of file [DS3231.h](#).

7.8.2.6 month

```
uint8_t ds3231_data_t::month
```

Definition at line 43 of file [DS3231.h](#).

7.8.2.7 year

```
uint8_t ds3231_data_t::year
```

Definition at line 44 of file [DS3231.h](#).

7.8.2.8 century

```
bool ds3231_data_t::century
```

Definition at line 45 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 306 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

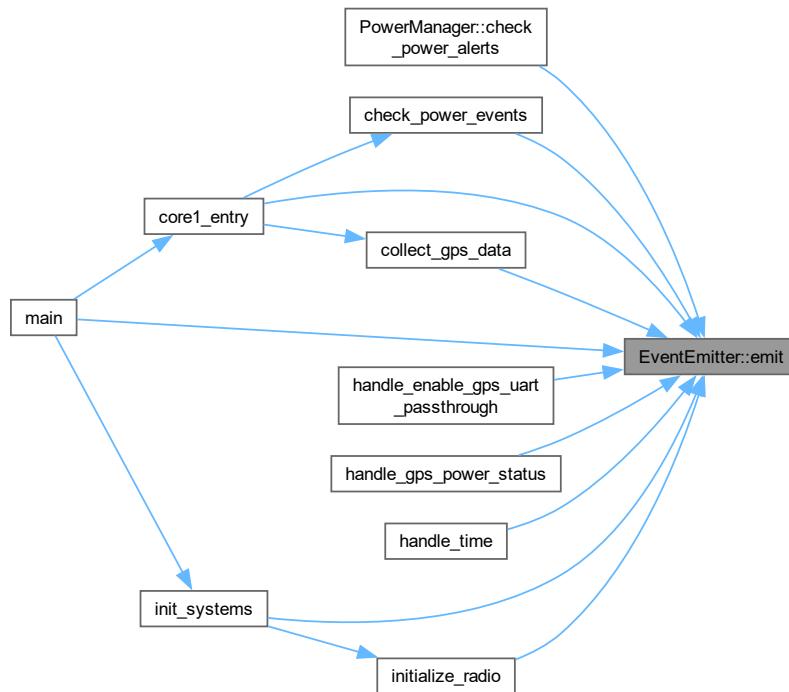
<i>group</i>	The event group.
<i>event</i>	The event identifier.

Template Parameters

<i>T</i>	The type of the event identifier.
----------	-----------------------------------

Definition at line 315 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 136 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 151 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 139 of file [event_manager.h](#).

7.10.3.2 timestamp

```
uint32_t EventLog::timestamp
```

Unix timestamp or system time.

Definition at line 141 of file [event_manager.h](#).

7.10.3.3 group

```
uint8_t EventLog::group
```

Event group identifier.

Definition at line 143 of file [event_manager.h](#).

7.10.3.4 event

```
uint8_t EventLog::event
```

Specific event identifier.

Definition at line 145 of file [event_manager.h](#).

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

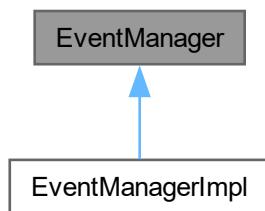
- lib/eventman/[event_manager.h](#)

7.11 EventManager Class Reference

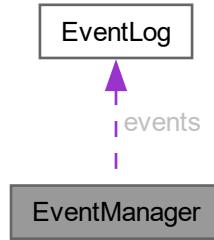
Manages the event logging system.

```
#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.
- `volatile uint16_t nextEventId`
Next event ID.
- `bool needsPersistence`
Flag indicating whether the events need to be saved to storage.

7.11.1 Detailed Description

Manages the event logging system.

Definition at line 165 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 171 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()

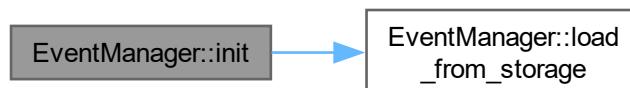
```
virtual void EventManager::init () [inline], [virtual]
```

Initializes the [EventManager](#).

Loads events from storage.

Definition at line 189 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 211 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 227 of file [event_manager.h](#).

7.11.4.2 eventCount

```
size_t EventManager::eventCount [protected]
```

Number of events in the buffer.

Definition at line 229 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 231 of file [event_manager.h](#).

7.11.4.4 eventMutex

```
mutex_t EventManager::eventMutex [protected]
```

Mutex for protecting the event buffer.

Definition at line 233 of file [event_manager.h](#).

7.11.4.5 nextEventId

```
volatile uint16_t EventManager::nextEventId [protected]
```

Next event ID.

Definition at line 235 of file [event_manager.h](#).

7.11.4.6 needsPersistence

```
bool EventManager::needsPersistence [protected]
```

Flag indicating whether the events need to be saved to storage.

Definition at line 237 of file [event_manager.h](#).

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

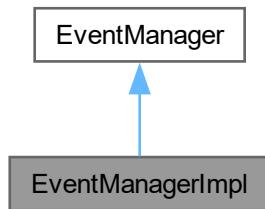
- lib/eventman/[event_manager.h](#)
- lib/eventman/[event_manager.cpp](#)

7.12 EventManagerImpl Class Reference

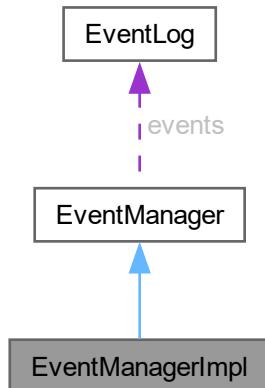
Implementation of the [EventManager](#) class.

```
#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.
- `volatile uint16_t nextEventId`
Next event ID.
- `bool needsPersistence`
Flag indicating whether the events need to be saved to storage.

7.12.1 Detailed Description

Implementation of the `EventManager` class.

Definition at line 245 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 251 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

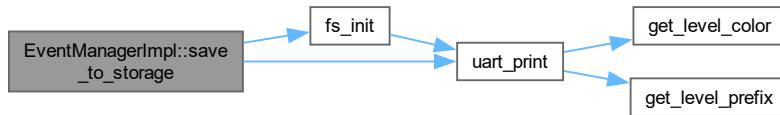
True if the events were successfully saved, false otherwise.

This method is not yet implemented.

Implements [EventManager](#).

Definition at line 260 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 290 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 209 of file [protocol.h](#).

7.14.2 Member Data Documentation

7.14.2.1 header

```
std::string Frame::header
```

Definition at line 210 of file [protocol.h](#).

7.14.2.2 direction

```
uint8_t Frame::direction
```

Definition at line 211 of file [protocol.h](#).

7.14.2.3 operationType

```
OperationType Frame::operationType
```

Definition at line 212 of file [protocol.h](#).

7.14.2.4 group

```
uint8_t Frame::group
```

Definition at line 213 of file [protocol.h](#).

7.14.2.5 command

```
uint8_t Frame::command
```

Definition at line 214 of file [protocol.h](#).

7.14.2.6 value

```
std::string Frame::value
```

Definition at line 215 of file [protocol.h](#).

7.14.2.7 unit

```
std::string Frame::unit
```

Definition at line 216 of file [protocol.h](#).

7.14.2.8 footer

```
std::string Frame::footer
```

Definition at line 217 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](#).

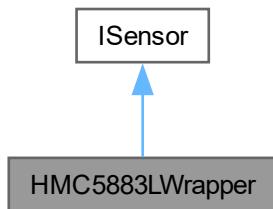
The documentation for this class was generated from the following files:

- lib/sensors/HMC5883L/[HMC5883L.h](#)
- lib/sensors/HMC5883L/[HMC5883L.cpp](#)

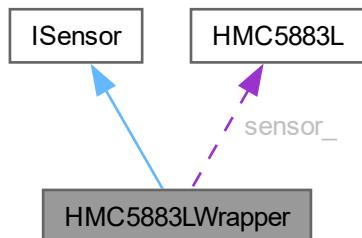
7.16 HMC5883LWrapper Class Reference

```
#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

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](#).

7.16.4 Member Data Documentation

7.16.4.1 `sensor_`

```
HMC5883L HMC5883LWrapper::sensor_ [private]
```

Definition at line 17 of file [HMC5883L_WRAPPER.h](#).

7.16.4.2 `initialized_`

```
bool HMC5883LWrapper::initialized_ [private]
```

Definition at line 18 of file [HMC5883L_WRAPPER.h](#).

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

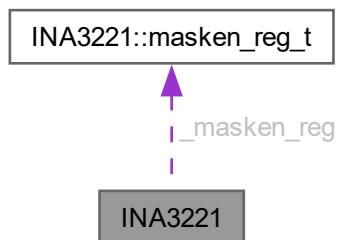
- lib/sensors/HMC5883L/[HMC5883L_WRAPPER.h](#)
- lib/sensors/HMC5883L/[HMC5883L_WRAPPER.cpp](#)

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]
```

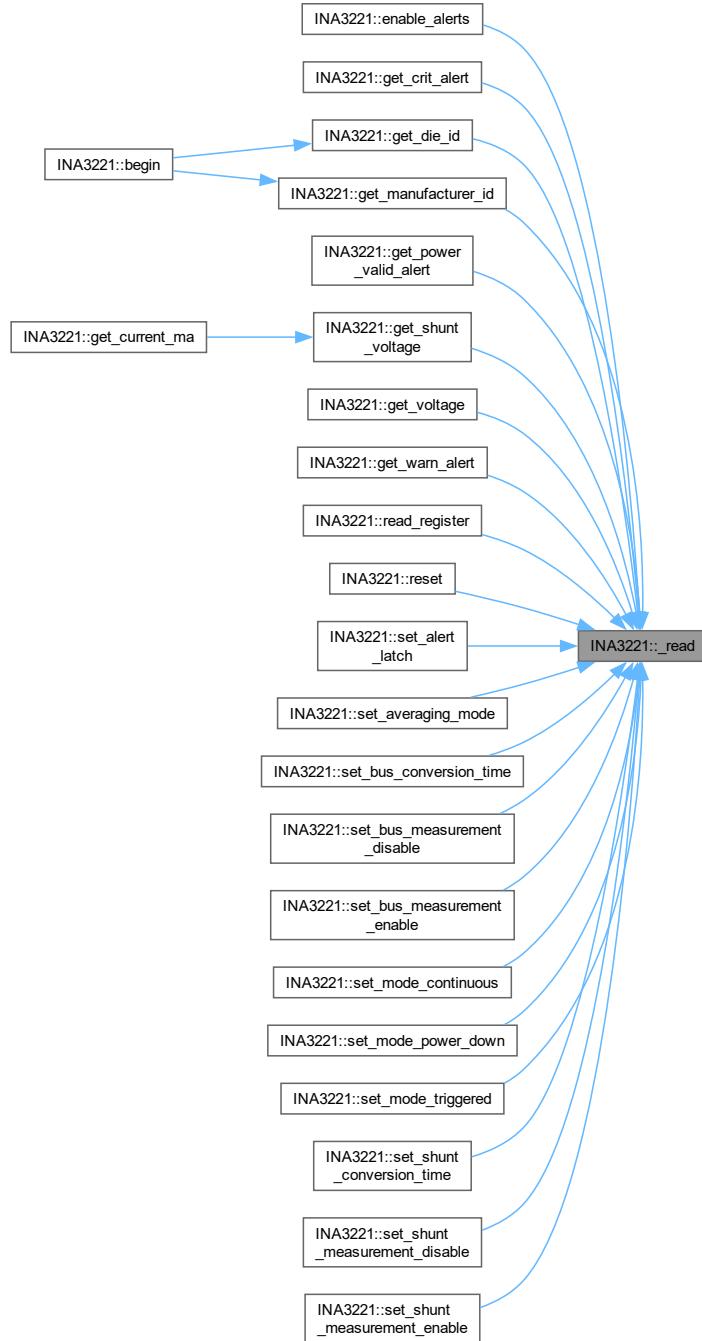
Read a 16-bit register from the device.

Parameters

<i>reg</i>	Register address
<i>val</i>	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]
```

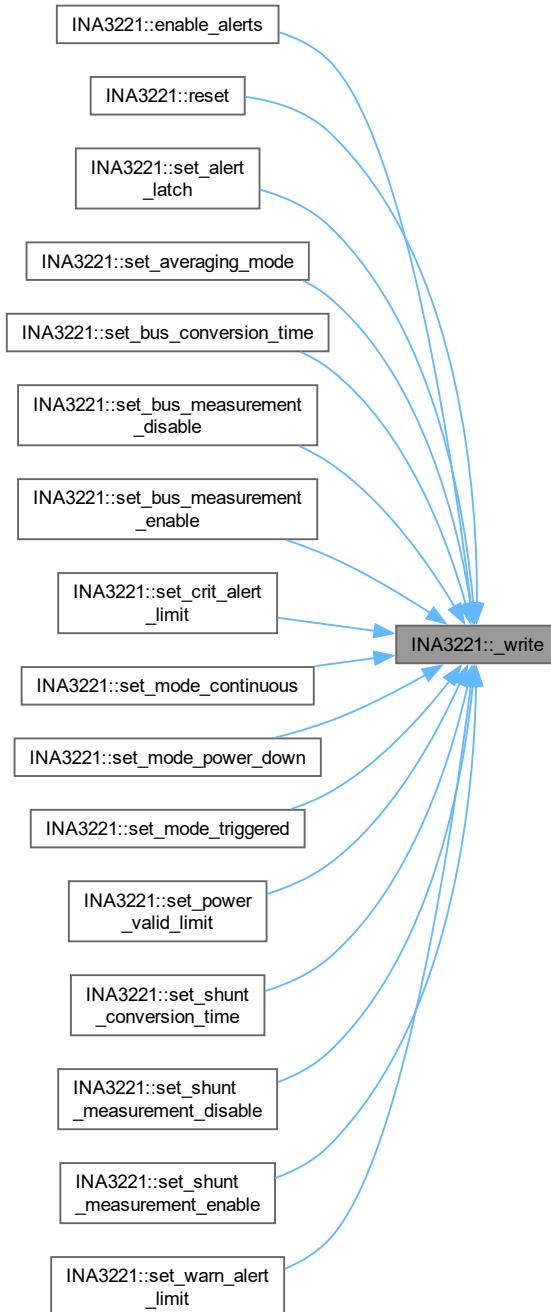
Write a 16-bit value to a register.

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](#).

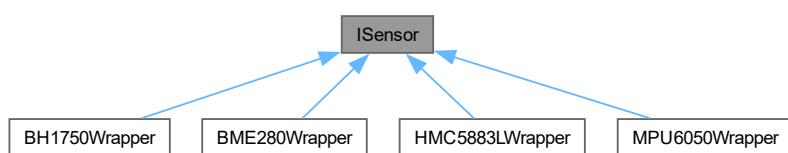
The documentation for this class was generated from the following files:

- 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 (SensorDataTypeIdentifier 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`

7.18.1 Detailed Description

Definition at line 33 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 (
    SensorDataTypeIdentifier 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](#).

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](#).

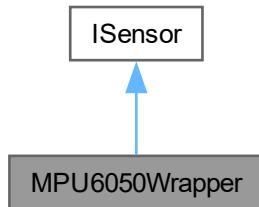
The documentation for this struct was generated from the following file:

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

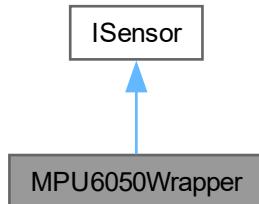
7.20 MPU6050Wrapper Class Reference

```
#include <MPU6050_WRAPPER.h>
```

Inheritance diagram for MPU6050Wrapper:



Collaboration diagram for MPU6050Wrapper:



Public Member Functions

- [MPU6050Wrapper \(\)](#)
- bool [init \(\)](#) override
- float [read_data \(SensorDataTypIdentifier type\)](#) 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](#)

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

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 9 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](#).

7.21.4 Member Data Documentation

7.21.4.1 rmc_tokens_

```
std::vector<std::string> NMEAData::rmc_tokens_ [private]
```

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

7.21.4.2 gga_tokens_

```
std::vector<std::string> NMEAData::gga_tokens_ [private]
```

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

7.21.4.3 rmc_mutex_

```
mutex_t NMEAData::rmc_mutex_ [private]
```

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

7.21.4.4 gga_mutex_

```
mutex_t NMEAData::gga_mutex_ [private]
```

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

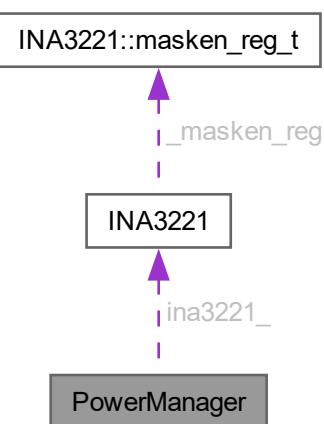
The documentation for this class was generated from the following files:

- 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 ()
```

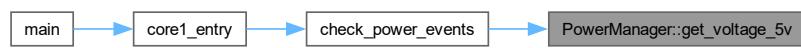
Definition at line 42 of file [PowerManager.cpp](#).

7.22.3.8 `get_voltage_5v()`

```
float PowerManager::get_voltage_5v ()
```

Definition at line 50 of file [PowerManager.cpp](#).

Here is the caller graph for this function:



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:



Here is the caller graph for this function:



7.22.3.11 is_charging_usb()

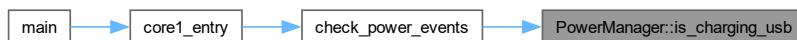
```
bool PowerManager::is_charging_usb ()
```

Definition at line 127 of file [PowerManager.cpp](#).

Here is the call graph for this function:



Here is the caller 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.

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 43 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]
```

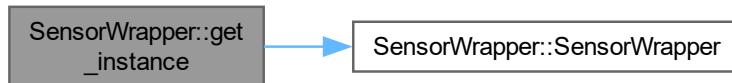
Provides a global instance of [SensorWrapper](#).

Returns

A reference to the single [SensorWrapper](#) instance.

Definition at line 23 of file [ISensor.cpp](#).

Here is the call 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

<code>type</code>	The sensor type (LIGHT, ENVIRONMENT, etc.).
<code>i2c</code>	The I2C interface pointer.

Returns

True if initialization succeeded, otherwise false.

Definition at line 39 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 63 of file [ISensor.cpp](#).

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

<i>sensorType</i>	The sensor type.
<i>dataType</i>	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 78 of file [ISensor.cpp](#).

7.23.4 Member Data Documentation**7.23.4.1 sensors**

```
std::map<SensorType, ISensor*> SensorWrapper::sensors [private]
```

Definition at line 51 of file [ISensor.h](#).

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

- lib/sensors/[ISensor.h](#)
- lib/sensors/[ISensor.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 380`

8.1.1 Macro Definition Documentation

8.1.1.1 BUILD_NUMBER

```
#define BUILD_NUMBER 380
```

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 380
00007
00008 #endif
```

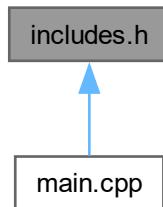
8.3 includes.h File Reference

```
#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 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
00036 #endif

```

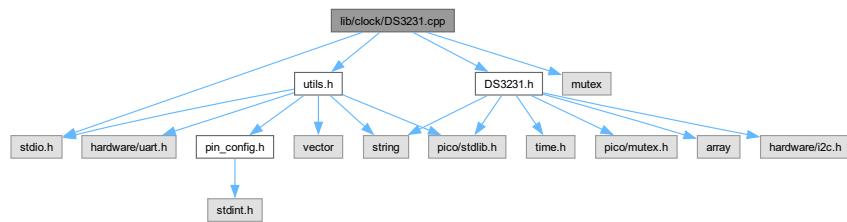
8.5 lib/clock/DS3231.cpp File Reference

```

#include "DS3231.h"
#include "utils.h"
#include <stdio.h>
#include <mutex>

```

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> // Include for printf
00004 #include <mutex> // Include for mutex
00005
00006 DS3231::DS3231(i2c_inst_t *i2c_instance) : i2c(i2c_instance), ds3231_addr(DS3231_DEVICE_ADDRESS) {
00007     // Initialize mutex (assuming you have a mutex member variable)
00008     recursive_mutex_init(&clock_mutex);
00009 }
00010
00011 int DS3231::set_time(ds3231_data_t *data) {
00012     uint8_t temp[7] = {0};
00013
00014     // Enable oscillator
00015     if (clock_enable() != 0) {
00016         uart_print("Failed to enable clock oscillator", VerbosityLevel::ERROR);
00017         return -1;
00018     }
00019
00020     if (data->seconds > 59)
00021         data->seconds = 59;
00022     if (data->minutes > 59)
00023         data->minutes = 59;
00024     if (data->hours > 23)
00025         data->hours = 23;
00026     if (data->day > 7)
00027         data->day = 7;
00028     else if (data->day < 1)
00029         data->day = 1;
00030     if (data->date > 31)
00031         data->date = 31;
00032     else if (data->date < 1)
00033         data->date = 1;
00034     if (data->month > 12)
00035         data->month = 12;
00036     else if (data->month < 1)
00037         data->month = 1;
00038     if (data->year > 99)
00039         data->year = 99;
00040
00041     temp[0] = bin_to_bcd(data->seconds);
00042     temp[1] = bin_to_bcd(data->minutes);
00043     temp[2] = bin_to_bcd(data->hours);
00044     temp[2] &= ~(0x01 « 6); // Clear 12/24 hour bit
00045     temp[3] = bin_to_bcd(data->day);
00046     temp[4] = bin_to_bcd(data->date);
00047     temp[5] = bin_to_bcd(data->month);
00048     if (data->century)
00049         temp[5] |= (0x01 « 7);
00050     temp[6] = bin_to_bcd(data->year);
00051
00052     std::string status = "BCD values to be written to DS3231: " + std::to_string(temp[0]) + " " +
00053                     std::to_string(temp[1]) + " " + std::to_string(temp[2]) + " " +
00054                     std::to_string(temp[3]) + " " + std::to_string(temp[4]) + " " +
00055                     std::to_string(temp[5]) + " " + std::to_string(temp[6]);
00056
00057     uart_print(status, VerbosityLevel::DEBUG);
00058
00059     int result = i2c_write_reg(DS3231_SECONDS_REG, 7, temp);
00060     if (result != 0) {
00061         uart_print("i2c write failed", VerbosityLevel::ERROR);
00062         return -1;
00063     }
00064
00065     return 0;
00066 }
00067
00068 int DS3231::get_time(ds3231_data_t *data) {
00069     std::string status;
00070     uint8_t raw_data[7];
00071     int result = i2c_read_reg(DS3231_SECONDS_REG, 7, raw_data);
00072     if (result != 0) {
00073         status = "Failed to read time from DS3231";
00074         uart_print(status, VerbosityLevel::ERROR);
00075         return -1;
00076     }
00077
00078     status = "Raw BCD values read from DS3231: " + std::to_string(raw_data[0]) + " " +
00079             std::to_string(raw_data[1]) + " " + std::to_string(raw_data[2]) + " " +
00080             std::to_string(raw_data[3]) + " " + std::to_string(raw_data[4]) + " " +
00081             std::to_string(raw_data[5]) + " " + std::to_string(raw_data[6]);
00082     uart_print(status, VerbosityLevel::DEBUG);
00083
00084     data->seconds = bcd_to_bin(raw_data[0] & 0x7F); // Masking for CH bit (clock halt)
00085     data->minutes = bcd_to_bin(raw_data[1] & 0x7F);
00086     data->hours = bcd_to_bin(raw_data[2] & 0x3F); // Masking for 12/24 hour mode bit
00087     data->day = raw_data[3] & 0x07; // Day of week (1-7)
00088     data->date = bcd_to_bin(raw_data[4] & 0x3F);
00089     data->month = bcd_to_bin(raw_data[5] & 0x1F); // Masking for century bit

```

```
00090     data->century = (raw_data[5] & 0x80) >> 7;
00091     data->year = bcd_to_bin(raw_data[6]);
00092
00093     // Data validation
00094     if (data->seconds > 59 || data->minutes > 59 || data->hours > 23 ||
00095         data->day < 1 || data->day > 7 || data->date < 1 || data->date > 31 ||
00096         data->month < 1 || data->month > 12 || data->year > 99) {
00097         uart_print("Invalid data read from DS3231", VerbosityLevel::ERROR);
00098         return -1;
00099     }
00100
00101     uart_print("Reading time from DS3231", VerbosityLevel::DEBUG);
00102     std::string timeStr = "Time: " + std::to_string(data->hours) + ":" + std::to_string(data->minutes)
00103     + ":" + std::to_string(data->seconds);
00104     uart_print(timeStr, VerbosityLevel::DEBUG);
00105     std::string dateStr = "Date: " + std::to_string(data->date) + "/" + std::to_string(data->month) +
00106     "/" + std::to_string(data->year);
00107     uart_print(dateStr, VerbosityLevel::DEBUG);
00108
00109     return 0;
00110 }
00111
00112 int DS3231::read_temperature(float *resolution) {
00113     std::string status;
00114     uint8_t temp[2];
00115     int result = i2c_read_reg(DS3231_TEMPERATURE_MSB_REG, 2, temp);
00116     if (result != 0) {
00117         status = "Failed to read temperature from DS3231";
00118         uart_print(status, VerbosityLevel::ERROR);
00119         return -1;
00120     }
00121     int8_t temperature_msb = (int8_t)temp[0]; // Signed for negative temperatures
00122     uint8_t temperature_lsb = temp[1] >> 6; // Only the 2 MSB are valid
00123     *resolution = temperature_msb + (temperature_lsb * 0.25f); // 0.25 degree resolution
00124
00125     return 0;
00126 }
00127
00128 int DS3231::i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
00129     if (!length)
00130         return -1;
00131
00132     std::string status = "Reading register " + std::to_string(reg_addr) + " from DS3231";
00133     uart_print(status, VerbosityLevel::DEBUG);
00134     recursive_mutex_enter_blocking(&clock_mutex_);
00135     uint8_t reg = reg_addr;
00136     int write_result = i2c_write_blocking(i2c, ds3231_addr, &reg, 1, true);
00137     if (write_result == PICO_ERROR_GENERIC) {
00138         status = "Failed to write register address to DS3231";
00139         uart_print(status, VerbosityLevel::ERROR);
00140         recursive_mutex_exit(&clock_mutex_);
00141         return -1;
00142     }
00143     int read_result = i2c_read_blocking(i2c, ds3231_addr, data, length, false);
00144     if (read_result == PICO_ERROR_GENERIC) {
00145         status = "Failed to read register data from DS3231";
00146         uart_print(status, VerbosityLevel::ERROR);
00147         recursive_mutex_exit(&clock_mutex_);
00148         return -1;
00149     }
00150     recursive_mutex_exit(&clock_mutex_);
00151
00152     return 0;
00153 }
00154
00155 int DS3231::i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
00156     if (!length)
00157         return -1;
00158
00159     recursive_mutex_enter_blocking(&clock_mutex_);
00160     uint8_t message[length + 1];
00161     message[0] = reg_addr;
00162     for (int i = 0; i < length; i++) {
00163         message[i + 1] = data[i];
00164     }
00165     int write_result = i2c_write_blocking(i2c, ds3231_addr, message, (length + 1), false);
00166     if (write_result == PICO_ERROR_GENERIC) {
00167         uart_print("Error: i2c_write_blocking failed in i2c_write_reg", VerbosityLevel::ERROR);
00168         recursive_mutex_exit(&clock_mutex_);
00169         return -1;
00170     }
00171     recursive_mutex_exit(&clock_mutex_);
00172
00173     return 0;
00174 }
```

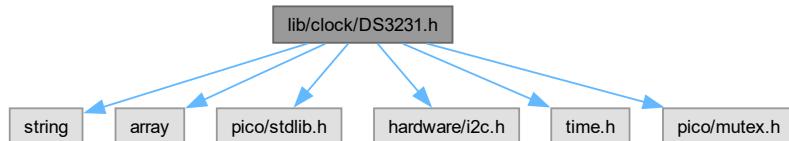
```

00191
00199 uint8_t DS3231::bin_to_bcd(const uint8_t data) {
00200     uint8_t ones_digit = (uint8_t)(data % 10);
00201     uint8_t tens_digit = (uint8_t)(data - ones_digit) / 10;
00202     return ((tens_digit << 4) + ones_digit);
00203 }
00204
00211 uint8_t DS3231::bcd_to_bin(const uint8_t bcd) {
00212     uint8_t ones_digit = (uint8_t)(bcd & 0x0F);
00213     uint8_t tens_digit = (uint8_t)(bcd >> 4);
00214     return (tens_digit * 10 + ones_digit);
00215 }
00216
00217 int DS3231::set_unix_time(time_t unix_time) {
00218     struct tm *timeinfo = gmtime(&unix_time);
00219     if (timeinfo == NULL) {
00220         uart_print("Error: gmtime() failed", VerbosityLevel::ERROR);
00221         return -1;
00222     }
00223
00224     ds3231_data_t data;
00225     data.seconds = timeinfo->tm_sec;
00226     data.minutes = timeinfo->tm_min;
00227     data.hours = timeinfo->tm_hour;
00228     data.day = timeinfo->tm_wday == 0 ? 7 : timeinfo->tm_wday; // Sunday is 0 in tm struct, but 1 in
00229     DS3231
00230     data.date = timeinfo->tm_mday;
00231     data.month = timeinfo->tm_mon + 1; // Month is 0-11 in tm struct, but 1-12 in DS3231
00232     data.year = timeinfo->tm_year - 100; // Year is since 1900, we want the last two digits
00233     data.century = timeinfo->tm_year >= 2000;
00234
00235     return set_time(&data);
00236 }
00237 time_t DS3231::get_unix_time() {
00238     ds3231_data_t data;
00239     if (get_time(&data)) {
00240         return -1; // Indicate error
00241     }
00242
00243     struct tm timeinfo;
00244     timeinfo.tm_sec = data.seconds;
00245     timeinfo.tm_min = data.minutes;
00246     timeinfo.tm_hour = data.hours;
00247     timeinfo.tm_mday = data.date;
00248     timeinfo.tm_mon = data.month - 1; // Month is 0-11 in tm struct, but 1-12 in DS3231
00249     timeinfo.tm_year = data.year + 100; // Year is since 1900
00250
00251     // mktime assumes that tm_wday and tm_yday are uninitialized
00252     timeinfo.tm_wday = 0;
00253     timeinfo.tm_yday = 0;
00254     timeinfo.tm_isdst = 0; // Set to 0 to use UTC
00255
00256     time_t timestamp = mktime(&timeinfo);
00257     if (timestamp == (time_t)(-1)) {
00258         uart_print("Error: mktime() failed", VerbosityLevel::ERROR);
00259         return -1;
00260     }
00261
00262     return timestamp;
00263 }
00264
00265 int DS3231::clock_enable() {
00266     std::string status;
00267     uint8_t control_reg = 0;
00268     int result = i2c_read_reg(DS3231_CONTROL_REG, 1, &control_reg);
00269     if (result != 0) {
00270         status = "Failed to read control register";
00271         uart_print(status, VerbosityLevel::ERROR);
00272         return -1;
00273     }
00274
00275     // Clear the EOSC bit to enable the oscillator
00276     control_reg &= ~(1 << 7);
00277
00278     result = i2c_write_reg(DS3231_CONTROL_REG, 1, &control_reg);
00279     if (result != 0) {
00280         status = "Failed to write control register";
00281         uart_print(status, VerbosityLevel::ERROR);
00282         return -1;
00283     }
00284
00285     return 0;
00286 }

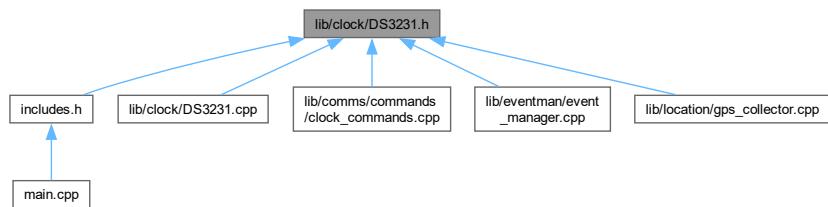
```

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 dependency graph for DS3231.h:
```



This graph shows which files directly or indirectly include this file:



Classes

- struct `ds3231_data_t`
- class `DS3231`

Macros

- `#define DS3231_DEVICE_ADDRESS 0x68`
- `#define DS3231_SECONDS_REG 0x00`
- `#define DS3231_MINUTES_REG 0x01`
- `#define DS3231_HOURS_REG 0x02`
- `#define DS3231_DAY_REG 0x03`
- `#define DS3231_DATE_REG 0x04`
- `#define DS3231_MONTH_REG 0x05`
- `#define DS3231_YEAR_REG 0x06`
- `#define DS3231_CONTROL_REG 0x0E`
- `#define DS3231_CONTROL_STATUS_REG 0x0F`
- `#define DS3231_TEMPERATURE_MSB_REG 0x11`
- `#define DS3231_TEMPERATURE_LSB_REG 0x12`

Enumerations

- enum days_of_week {
 MONDAY = 1 , TUESDAY , WEDNESDAY , THURSDAY ,
 FRIDAY , SATURDAY , SUNDAY }

8.7.1 Macro Definition Documentation

8.7.1.1 DS3231_DEVICE_ADDRESS

```
#define DS3231_DEVICE_ADDRESS 0x68
```

Definition at line 11 of file [DS3231.h](#).

8.7.1.2 DS3231_SECONDS_REG

```
#define DS3231_SECONDS_REG 0x00
```

Definition at line 13 of file [DS3231.h](#).

8.7.1.3 DS3231_MINUTES_REG

```
#define DS3231_MINUTES_REG 0x01
```

Definition at line 14 of file [DS3231.h](#).

8.7.1.4 DS3231_HOURS_REG

```
#define DS3231_HOURS_REG 0x02
```

Definition at line 15 of file [DS3231.h](#).

8.7.1.5 DS3231_DAY_REG

```
#define DS3231_DAY_REG 0x03
```

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

8.7.1.6 DS3231_DATE_REG

```
#define DS3231_DATE_REG 0x04
```

Definition at line 17 of file [DS3231.h](#).

8.7.1.7 DS3231_MONTH_REG

```
#define DS3231_MONTH_REG 0x05
```

Definition at line 18 of file [DS3231.h](#).

8.7.1.8 DS3231_YEAR_REG

```
#define DS3231_YEAR_REG 0x06
```

Definition at line 19 of file [DS3231.h](#).

8.7.1.9 DS3231_CONTROL_REG

```
#define DS3231_CONTROL_REG 0x0E
```

Definition at line 21 of file [DS3231.h](#).

8.7.1.10 DS3231_CONTROL_STATUS_REG

```
#define DS3231_CONTROL_STATUS_REG 0x0F
```

Definition at line 22 of file [DS3231.h](#).

8.7.1.11 DS3231_TEMPERATURE_MSB_REG

```
#define DS3231_TEMPERATURE_MSB_REG 0x11
```

Definition at line 24 of file [DS3231.h](#).

8.7.1.12 DS3231_TEMPERATURE_LSB_REG

```
#define DS3231_TEMPERATURE_LSB_REG 0x12
```

Definition at line 25 of file [DS3231.h](#).

8.7.2 Enumeration Type Documentation

8.7.2.1 days_of_week

```
enum days_of_week
```

Enumerator

MONDAY	
TUESDAY	
WEDNESDAY	
THURSDAY	
FRIDAY	
SATURDAY	
SUNDAY	

Definition at line 27 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
00011 #define DS3231_DEVICE_ADDRESS          0x68
00012
00013 #define DS3231_SECONDS_REG           0x00
00014 #define DS3231_MINUTES_REG          0x01
00015 #define DS3231_HOURS_REG            0x02
00016 #define DS3231_DAY_REG              0x03
00017 #define DS3231_DATE_REG             0x04
00018 #define DS3231_MONTH_REG            0x05
00019 #define DS3231_YEAR_REG             0x06
00020
00021 #define DS3231_CONTROL_REG          0x0E
00022 #define DS3231_CONTROL_STATUS_REG   0x0F
00023
00024 #define DS3231_TEMPERATURE_MSB_REG  0x11
00025 #define DS3231_TEMPERATURE_LSB_REG  0x12
00026
00027 enum days_of_week {
00028     MONDAY    = 1,
00029     TUESDAY,
00030     WEDNESDAY,
00031     THURSDAY,
00032     FRIDAY,
00033     SATURDAY,
00034     SUNDAY
00035 };
00036
00037 typedef struct {
00038     uint8_t seconds;
00039     uint8_t minutes;
00040     uint8_t hours;
00041     uint8_t day;
00042     uint8_t date;
00043     uint8_t month;
00044     uint8_t year;
00045     bool century;
00046 } ds3231_data_t;
00047
00048 class DS3231 {
00049 public:
00050     DS3231(i2c_inst_t *i2c_instance);
00051
00052     int set_time(ds3231_data_t *data);
00053     int get_time(ds3231_data_t *data);
00054     int read_temperature(float *resolution);
00055
00056     int set_unix_time(time_t unix_time);
00057     time_t get_unix_time();
00058     int clock_enable();
00059
00060 private:
00061     i2c_inst_t *i2c;
00062     uint8_t ds3231_addr;
00063
00064     int i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data);
00065     int i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data);
00066
00067     uint8_t bin_to_bcd(const uint8_t data);
00068     uint8_t bcd_to_bin(const uint8_t bcd);
00069
00070     recursive_mutex_t clock_mutex; // Mutex for I2C access
00071 };
00072
00073 #endif

```

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 ¶m, OperationType operationType)`
Handler for getting and setting system time.
- `std::vector< Frame > handle_timezone_offset (const std::string ¶m, OperationType operationType)`
Handler for getting and setting timezone offset.
- `std::vector< Frame > handle_clock_sync_interval (const std::string ¶m, OperationType operationType)`
Handler for getting and setting clock synchronization interval.
- `std::vector< Frame > handle_get_last_sync_time (const std::string ¶m, 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
00010
00016
00017 extern DS3231 systemClock;
00018
00032 std::vector<Frame> handle_time(const std::string& param, OperationType operationType) {
00033     std::vector<Frame> frames;
00034
00035     if (operationType == OperationType::SET) {
00036         if (param.empty()) {
00037             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, "PARAM REQUIRED"));
00038             return frames;
00039         }
00040         try {
00041             time_t newTime = std::stoll(param);
00042             if (newTime <= 0) {
00043                 frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, "TIME MUST BE
00044                 POSITIVE"));
00045                 return frames;
00046             }
00047             if (systemClock.set_unix_time(newTime) != 0) {
00048                 frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, "FAILED TO SET
00049                 TIME"));
00050             }
00051
00052             EventEmitter::emit(EventGroup::CLOCK, ClockEvent::CHANGED);
00053             frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, TIME,
00054                 std::to_string(systemClock.get_unix_time())));
00055             return frames;
00056         } catch (...) {
00057             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, "INVALID TIME
00058             FORMAT"));
00059         }
00060     } else if (operationType == OperationType::GET) {
00061         if (!param.empty()) {
```

```
00061         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, "PARAM UNNECESSARY"));
00062         return frames;
00063     }
00064
00065     uint32_t timeUnix = systemClock.get_unix_time();
00066     if (timeUnix == 0) {
00067         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, "FAILED TO GET
TIME"));
00068         return frames;
00069     }
00070
00071     frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, TIME,
std::to_string(timeUnix)));
00072     return frames;
00073 }
00074
00075     frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, "INVALID OPERATION"));
00076     return frames;
00077 }
00078
00079 std::vector<Frame> handle_timezone_offset(const std::string& param, OperationType operationType) {
00080     std::vector<Frame> frames;
00081
00082     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00083         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID
OPERATION"));
00084         return frames;
00085     }
00086
00087     if (operationType == OperationType::GET) {
00088         if (!param.empty()) {
00089             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, "PARAM
UNNECESSARY"));
00090             return frames;
00091         }
00092
00093         frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, TIMEZONE_OFFSET, "60"));
00094         return frames;
00095     }
00096
00097     if (param.empty()) {
00098         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, "PARAM
REQUIRED"));
00099         return frames;
00100     }
00101
00102     try {
00103         int16_t offset = std::stoi(param);
00104         if (offset < -720 || offset > 720) {
00105             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID
OFFSET"));
00106             return frames;
00107         }
00108
00109         frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, TIMEZONE_OFFSET, param));
00110         return frames;
00111     } catch (...) {
00112         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID
PARAMETER"));
00113         return frames;
00114     }
00115 }
00116
00117
00118
00119 std::vector<Frame> handle_clock_sync_interval(const std::string& param, OperationType operationType) {
00120     std::vector<Frame> frames;
00121
00122     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00123         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "INVALID
OPERATION"));
00124         return frames;
00125     }
00126
00127
00128     if (operationType == OperationType::GET) {
00129         if (!param.empty()) {
00130             frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "PARAM
UNNECESSARY"));
00131             return frames;
00132         }
00133
00134         std::string clockSyncInterval = "1440";
00135         frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
clockSyncInterval));
00136         return frames;
00137     }
00138
00139     if (operationType == OperationType::SET) {
```

```

00158     if (param.empty()) {
00159         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "PARAM
REQUIRED"));
00160     }
00161     try {
00162         uint32_t interval = std::stoul(param);
00163         //set sync interval
00164
00165         frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00166             param));
00167         return frames;
00168     } catch (...) {
00169         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
00170             "INVALID PARAMETER"));
00171     }
00172 }
00173 frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "UNKNOWN
ERROR"));
00174 return frames;
00175 }
00176
00186 std::vector<Frame> handle_get_last_sync_time(const std::string& param, OperationType operationType) {
00187     std::vector<Frame> frames;
00188     if (operationType != OperationType::GET || !param.empty()) {
00189         frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, LAST_SYNC_TIME, "INVALID
REQUEST"));
00190         return frames;
00191     }
00192     std::string lastSyncTime = "none";
00193     frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, LAST_SYNC_TIME, lastSyncTime));
00194     return frames;
00195 } // 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 ¶m, OperationType operationType)
Executes a command based on its key.

Variables

- **CommandMap commandHandlers**

Global map of all command handlers.

8.12 commands.cpp

[Go to the documentation of this file.](#)

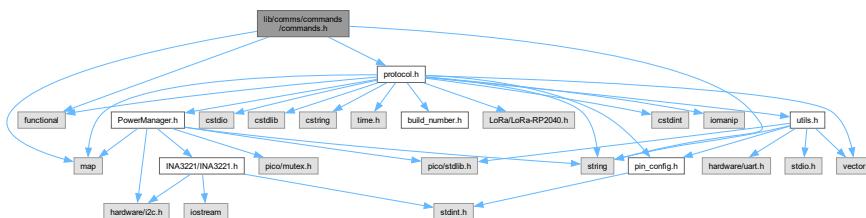
```

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 commandHandlers = {
00028     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(0)), handle_get_commands_list},
00029     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(1)), handle_get_build_version},
00030     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(8)), handle_verbosity},
00031     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(9)), handle_enter_bootloader_mode},
00032     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(0)), handle_get_power_manager_ids},
00033     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(2)), handle_get_voltage_battery},
00034     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(3)), handle_get_voltage_5v},
00035     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(4)), handle_get_current_charge_usb},
00036     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(5)), handle_get_current_charge_solar},
00037     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(6)), handle_get_current_charge_total},
00038     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(7)), handle_get_current_draw},
00039     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(0)), handle_time},
00040     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(1)), handle_timezone_offset},
00041     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(2)), handle_clock_sync_interval},
00042     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(3)), handle_get_last_sync_time},
00043     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(1)), handle_get_last_events},
00044     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(2)), handle_get_event_count},
00045     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(0)), handle_list_files},
00046     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(1)), handle_file_download},
00047     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(4)), handle_mount},
00048     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(1)), handle_gps_power_status},
00049     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(2)), handle_enable_gps_uart_passthrough},
00050     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(3)), handle_get_rmc_data},
00051     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(4)), handle_get_gga_data},
00052 };
00053
00054
00063 std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType
00064     operationType) {
00065     auto it = commandHandlers.find(commandKey);
00066     if (it != commandHandlers.end()) {
00067         CommandHandler handler = it->second;
00068     } else {
00069         std::vector<Frame> frames;
00070         frames.push_back(frame_build(OperationType::ERR, 0, 0, "INVALID COMMAND"));
00071     }
00072 }
00073 } // end of CommandSystem group

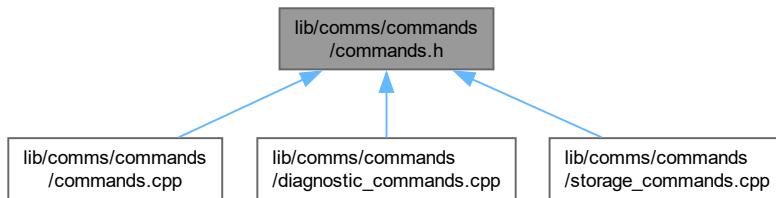
```

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 ¶m, OperationType operationType)`
Handler for getting and setting system time.
- `std::vector< Frame > handle_timezone_offset (const std::string ¶m, OperationType operationType)`
Handler for getting and setting timezone offset.
- `std::vector< Frame > handle_clock_sync_interval (const std::string ¶m, OperationType operationType)`
Handler for getting and setting clock synchronization interval.
- `std::vector< Frame > handle_get_last_sync_time (const std::string ¶m, OperationType operationType)`
Handler for getting last clock sync time.
- `std::vector< Frame > handle_get_commands_list (const std::string ¶m, OperationType operationType)`
Handler for listing all available commands on UART.
- `std::vector< Frame > handle_get_build_version (const std::string ¶m, OperationType operationType)`
Get firmware build version.
- `std::vector< Frame > handle_verbosity (const std::string ¶m, OperationType operationType)`
Handles setting or getting the UART verbosity level.
- `std::vector< Frame > handle_enter_bootloader_mode (const std::string ¶m, OperationType operationType)`
Reboot system to USB firmware loader.

- std::vector< Frame > handle_gps_power_status (const std::string ¶m, OperationType operationType)
Handler for controlling GPS module power state.
- std::vector< Frame > handle_enable_gps_uart_passthrough (const std::string ¶m, OperationType operationType)
Handler for enabling GPS transparent mode (UART pass-through)
- std::vector< Frame > handle_get_rmc_data (const std::string ¶m, OperationType operationType)
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- std::vector< Frame > handle_get_gga_data (const std::string ¶m, OperationType operationType)
Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.
- std::vector< Frame > handle_get_power_manager_ids (const std::string ¶m, OperationType operationType)
Handler for retrieving Power Manager IDs.
- std::vector< Frame > handle_get_voltage_battery (const std::string ¶m, OperationType operationType)
Handler for getting battery voltage.
- std::vector< Frame > handle_get_voltage_5v (const std::string ¶m, OperationType operationType)
Handler for getting 5V rail voltage.
- std::vector< Frame > handle_get_current_charge_usb (const std::string ¶m, OperationType operationType)
Handler for getting USB charge current.
- std::vector< Frame > handle_get_current_charge_solar (const std::string ¶m, OperationType operationType)
Handler for getting solar panel charge current.
- std::vector< Frame > handle_get_current_charge_total (const std::string ¶m, OperationType operationType)
Handler for getting total charge current.
- std::vector< Frame > handle_get_current_draw (const std::string ¶m, OperationType operationType)
Handler for getting system current draw.
- std::vector< Frame > handle_get_last_events (const std::string ¶m, OperationType operationType)
Handler for retrieving last N events from the event log.
- std::vector< Frame > handle_get_event_count (const std::string ¶m, OperationType operationType)
Handler for getting total number of events in the log.
- std::vector< Frame > handle_list_files (const std::string ¶m, OperationType operationType)
Handles the list files command.
- std::vector< Frame > handle_file_download (const std::string ¶m, OperationType operationType)
Handles the file download command.
- std::vector< Frame > handle_mount (const std::string ¶m, OperationType operationType)
Handles the SD card mount/unmount command.
- std::vector< Frame > execute_command (uint32_t commandKey, const std::string ¶m, OperationType operationType)
Executes a command based on its key.

Variables

- std::map< uint32_t, std::function< std::vector< Frame > (const std::string &, OperationType)> > commandHandlers
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
00011 // CLOCK
00012 std::vector<Frame> handle_time(const std::string& param, OperationType operationType);
00013 std::vector<Frame> handle_timezone_offset(const std::string& param, OperationType operationType);
00014 std::vector<Frame> handle_clock_sync_interval(const std::string& param, OperationType operationType);
00015 std::vector<Frame> handle_get_last_sync_time(const std::string& param, OperationType operationType);
00016
00017
00018 // DIAG
00019 std::vector<Frame> handle_get_commands_list(const std::string& param, OperationType operationType);
00020 std::vector<Frame> handle_get_build_version(const std::string& param, OperationType operationType);
00021 std::vector<Frame> handle_verbosity(const std::string& param, OperationType operationType);
00022 std::vector<Frame> handle_enter_bootloader_mode(const std::string& param, OperationType
operationType);
00023
00024
00025 // GPS
00026 std::vector<Frame> handle_gps_power_status(const std::string& param, OperationType operationType);
00027 std::vector<Frame> handle_enable_gps_uart_passthrough(const std::string& param, OperationType
operationType);
00028 std::vector<Frame> handle_get_rmc_data(const std::string& param, OperationType operationType);
00029 std::vector<Frame> handle_get_gga_data(const std::string& param, OperationType operationType);
00030
00031
00032 // POWER
00033 std::vector<Frame> handle_get_power_manager_ids(const std::string& param, OperationType
operationType);
00034 std::vector<Frame> handle_get_voltage_battery(const std::string& param, OperationType operationType);
00035 std::vector<Frame> handle_get_voltage_5v(const std::string& param, OperationType operationType);
00036 std::vector<Frame> handle_get_current_charge_usb(const std::string& param, OperationType
operationType);
00037 std::vector<Frame> handle_get_current_charge_solar(const std::string& param, OperationType
operationType);
00038 std::vector<Frame> handle_get_current_charge_total(const std::string& param, OperationType
operationType);
00039 std::vector<Frame> handle_get_current_draw(const std::string& param, OperationType operationType);
00040
00041
00042 // EVENT
00043 std::vector<Frame> handle_get_last_events(const std::string& param, OperationType operationType);
00044 std::vector<Frame> handle_get_event_count(const std::string& param, OperationType operationType);
00045
00046
00047 //STORAGE
00048 std::vector<Frame> handle_list_files(const std::string& param, OperationType operationType);
00049 std::vector<Frame> handle_file_download(const std::string& param, OperationType operationType);
00050 std::vector<Frame> handle_mount(const std::string& param, OperationType operationType);
00051
00052 std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType
operationType);
00053 extern std::map<uint32_t, std::function<std::vector<Frame>(const std::string&, OperationType)>>
commandHandlers;
00054
00055 #endif

```

8.15 lib/comms/commands/diagnostic_commands.cpp File Reference

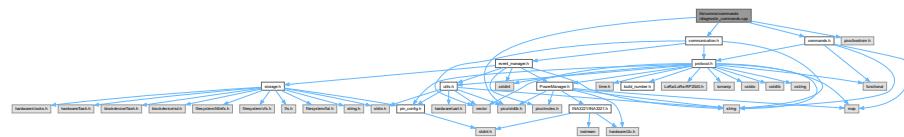
```

#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 ¶m, OperationType operationType)
Handler for listing all available commands on UART.
- std::vector< Frame > **handle_get_build_version** (const std::string ¶m, OperationType operationType)
Get firmware build version.
- std::vector< Frame > **handle_verbosity** (const std::string ¶m, OperationType operationType)
Handles setting or getting the UART verbosity level.
- std::vector< Frame > **handle_enter_bootloader_mode** (const std::string ¶m, 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
00026     if (!param.empty()) {
00027         frames.push_back(frame_build(OperationType::ERR, 1, 0, "PARAM_UNNECESSARY"));
00028         return frames;
00029     }
00030
00031     if (!(operationType == OperationType::GET)) {
00032         frames.push_back(frame_build(OperationType::ERR, 1, 0, "INVALID_OPERATION"));
00033         return frames;
00034     }
00035
00036     for (const auto& entry : commandHandlers) {
00037         uint32_t commandKey = entry.first;
00038         uint8_t group = (commandKey >> 8) & 0xFF;
00039         uint8_t command = commandKey & 0xFF;
00040
00041         std::string commandDetails = std::to_string(group) + "-" + std::to_string(command);
00042         frames.push_back(frame_build(OperationType::SEQ, 1, 0, commandDetails));
00043     }
00044
00045     // Add final VAL frame
00046     frames.push_back(frame_build(OperationType::VAL, 1, 0, "SEQ_DONE"));
00047     return frames;
00048 }
```

```

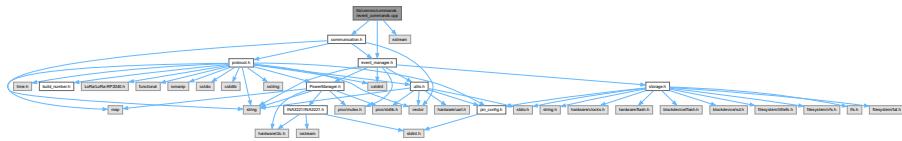
00049
00050
00061 std::vector<Frame> handle_get_build_version(const std::string& param, OperationType operationType) {
00062     std::vector<Frame> frames;
00063
00064     if (!param.empty()) {
00065         frames.push_back(frame_build(OperationType::ERR, 1, 1, "PARAM_UNNECESSARY"));
00066         return frames;
00067     }
00068
00069     if (operationType == OperationType::GET) {
00070         frames.push_back(frame_build(OperationType::VAL, 1, 1, std::to_string(BUILD_NUMBER)));
00071         return frames;
00072     }
00073
00074     frames.push_back(frame_build(OperationType::ERR, 1, 1, "INVALID_OPERATION"));
00075     return frames;
00076 }
00077
00078
00100 std::vector<Frame> handle_verbosity(const std::string& param, OperationType operationType) {
00101     std::vector<Frame> frames;
00102
00103     if (operationType == OperationType::GET && param.empty()) {
00104         uart_print("Current verbosity level: " + std::to_string(static_cast<int>(g_uart_verbosity)),
00105                     VerbosityLevel::INFO);
00106         frames.push_back(frame_build(OperationType::VAL, 1, 8,
00107                                 std::to_string(static_cast<int>(g_uart_verbosity))));
00108         return frames;
00109     }
00110
00111     try {
00112         int level = std::stoi(param);
00113         if (level < 0 || level > 5) {
00114             frames.push_back(frame_build(OperationType::ERR, 1, 8, "INVALID LEVEL (0-5")));
00115             return frames;
00116         }
00117         g_uart_verbosity = static_cast<VerbosityLevel>(level);
00118         frames.push_back(frame_build(OperationType::RES, 1, 8, "SET " + std::to_string(level)));
00119         return frames;
00120     } catch (...) {
00121         frames.push_back(frame_build(OperationType::ERR, 1, 8, "INVALID FORMAT"));
00122         return frames;
00123     }
00124 }
00125
00136 std::vector<Frame> handle_enter_bootloader_mode(const std::string& param, OperationType operationType)
{
00137     std::vector<Frame> frames;
00138
00139     if (param != "USB") {
00140         frames.push_back(frame_build(OperationType::ERR, 1, 9, "PARAM_INVALID"));
00141         return frames;
00142     }
00143
00144     if (operationType != OperationType::SET) {
00145         frames.push_back(frame_build(OperationType::ERR, 1, 9, "INVALID_OPERATION"));
00146         return frames;
00147     }
00148
00149     frames.push_back(frame_build(OperationType::RES, 1, 9, "REBOOT BOOTSEL"));
00150
00151     // Set flag to trigger reboot after function returns
00152     g_pending_bootloader_reset = true;
00153
00154     return frames;
00155 }
00156

```

8.17 lib/comms/commands/event_commands.cpp File Reference

```
#include "communication.h"
#include "event_manager.h"
#include <iostream>
```

Include dependency graph for event_commands.cpp:



Functions

- std::vector< Frame > handle_get_last_events (const std::string ¶m, OperationType operationType)
Handler for retrieving last N events from the event log.
- std::vector< Frame > handle_get_event_count (const std::string ¶m, 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 <iostream>
00004
00005
00011
00029 std::vector<Frame> handle_get_last_events(const std::string& param, OperationType operationType) {
00030     std::vector<Frame> frames;
00031     if (operationType != OperationType::GET) {
00032         frames.push_back(frame_build(OperationType::ERR, 5, 1, "INVALID OPERATION"));
00033         return frames;
00034     }
00035     size_t count = 10; // Default number of events to return
00036     if (!param.empty()) {
00037         try {
00038             count = std::stoul(param);
00039             if (count == 0 || count > EVENT_BUFFER_SIZE) {
00040                 frames.push_back(frame_build(OperationType::ERR, 5, 1, "INVALID COUNT"));
00041                 return frames;
00042             }
00043         } catch (...) {
00044             frames.push_back(frame_build(OperationType::ERR, 5, 1, "INVALID PARAMETER"));
00045             return frames;
00046         }
00047     }
00048
00049     std::stringstream ss;
00050     ss << std::hex << std::uppercase << std::setfill('0');
00051
00052     size_t available = eventManager.get_event_count();
00053     size_t toReturn = std::min(count, available);
00054
00055     // Start from the most recent event
00056     for (size_t i = 0; i < toReturn; i++) {
00057         const EventLog& event = eventManager.get_event(available - 1 - i);
00058         // Format: IIIITTTTGGEE
00059         // IIII: 16-bit ID (4 hex chars)
00060         // TTTTTTTT: 32-bit timestamp (8 hex chars)
00061         // GG: 8-bit group (2 hex chars)
00062         // EE: 8-bit event (2 hex chars)
00063         ss << std::setw(4) << event.id
00064             << std::setw(8) << event.timestamp
00065             << std::setw(2) << static_cast<int>(event.group)
00066             << std::setw(2) << static_cast<int>(event.event);
00067         if (i < toReturn - 1) ss << "-";
00068     }
00069     frames.push_back(frame_build(OperationType::VAL, 5, 1, ss.str()));
00070     return frames;
00071 }
00072
00073
00086 std::vector<Frame> handle_get_event_count(const std::string& param, OperationType operationType) {

```

```

00087     std::vector<Frame> frames;
00088     if (operationType != OperationType::GET || !param.empty()) {
00089         frames.push_back(frame_build(OperationType::ERR, 5, 2, "INVALID REQUEST"));
00090         return frames;
00091     }
00092     frames.push_back(frame_build(OperationType::VAL, 5, 2,
00093         std::to_string(eventManager.get_event_count())));
00093     return frames;
00094 } // 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:

```



Functions

- std::vector< Frame > **handle_gps_power_status** (const std::string ¶m, OperationType operationType)
Handler for controlling GPS module power state.
- std::vector< Frame > **handle_enable_gps_uart_passthrough** (const std::string ¶m, OperationType operationType)
Handler for enabling GPS transparent mode (UART pass-through)
- std::vector< Frame > **handle_get_rmc_data** (const std::string ¶m, OperationType operationType)
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- std::vector< Frame > **handle_get_gga_data** (const std::string ¶m, OperationType operationType)
Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.

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 <sstream> // Include for stringstream
00004
00010
00026 std::vector<Frame> handle_gps_power_status(const std::string& param, OperationType operationType) {
00027     std::vector<Frame> frames;
00028
00029     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00030         frames.push_back(frame_build(OperationType::ERR, 7, 1, "INVALID OPERATION"));
00031         return frames;
00032     }
00033
00034     if (operationType == OperationType::SET) {
00035         if (param.empty()) {
00036             frames.push_back(frame_build(OperationType::ERR, 7, 1, "PARAM REQUIRED"));
00037             return frames;
00038         }
00039
00040         try {
00041             int powerStatus = std::stoi(param);

```

```
00042         if (powerStatus != 0 && powerStatus != 1) {
00043             frames.push_back(frame_build(OperationType::ERR, 7, 1, "INVALID VALUE. USE 0 OR 1"));
00044             return frames;
00045         }
00046         gpio_put(GPS_POWER_ENABLE_PIN, powerStatus);
00047         EventEmitter::emit(EventGroup::GPS, powerStatus ? GPSEvent::POWER_ON :
GPSEvent::POWER_OFF);
00048         frames.push_back(frame_build(OperationType::RES, 7, 1, std::to_string(powerStatus)));
00049         return frames;
00050     } catch (...) {
00051         frames.push_back(frame_build(OperationType::ERR, 7, 1, "INVALID PARAMETER FORMAT"));
00052         return frames;
00053     }
00054 }
00055
00056 // GET operation
00057 if (!param.empty()) {
00058     frames.push_back(frame_build(OperationType::ERR, 7, 1, "PARAM UNNECESSARY"));
00059     return frames;
00060 }
00061
00062 bool powerStatus = gpio_get(GPS_POWER_ENABLE_PIN);
00063 frames.push_back(frame_build(OperationType::VAL, 7, 1, std::to_string(powerStatus)));
00064 return frames;
00065 }
00066
00067
00083 std::vector<Frame> handle_enable_gps_uart_passthrough(const std::string& param, OperationType
operationType) {
00084     std::vector<Frame> frames;
00085     // Validate operation type
00086     if (!(operationType == OperationType::SET)) {
00087         frames.push_back(frame_build(OperationType::ERR, 7, 2, "NOT ALLOWED"));
00088         return frames;
00089     }
00090
00091     // Parse and validate timeout parameter
00092     uint32_t timeoutMs;
00093     try {
00094         timeoutMs = param.empty() ? 60000u : std::stoul(param) * 1000;
00095     } catch (...) {
00096         frames.push_back(frame_build(OperationType::ERR, 7, 2, "INVALID TIMEOUT FORMAT"));
00097         return frames;
00098     }
00099
00100    // Setup UART parameters and exit sequence
00101    const std::string EXIT_SEQUENCE = "##EXIT##";
00102    std::string inputBuffer;
00103    bool exitRequested = false;
00104    uint32_t originalBaudRate = DEBUG_UART_BAUD_RATE;
00105    uint32_t gpsBaudRate = GPS_UART_BAUD_RATE;
00106    uint32_t startTime = to_ms_since_boot(get_absolute_time());
00107
00108    // Log start of transparent mode
00109    EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_START);
00110
00111    // Print startup message
00112    std::string message = "Entering GPS Serial Pass-Through Mode @" +
00113        std::to_string(gpsBaudRate) + " for " +
00114        std::to_string(timeoutMs/1000) + "s\r\n" +
00115        "Send " + EXIT_SEQUENCE + " to exit";
00116    uart_print(message, VerbosityLevel::INFO);
00117    // Allow time for message to be sent before baudrate change
00118    sleep_ms(10);
00119
00120    // Switch to GPS baudrate
00121    uart_set_baudrate(DEBUG_UART_PORT, gpsBaudRate);
00122
00123    // Main transparent mode loop
00124    while (!exitRequested) {
00125        while (uart_is_readable(DEBUG_UART_PORT)) {
00126            char ch = uart_getc(DEBUG_UART_PORT);
00127
00128            inputBuffer += ch;
00129            if (inputBuffer.length() > EXIT_SEQUENCE.length()) {
00130                inputBuffer = inputBuffer.substr(1);
00131            }
00132
00133            if (inputBuffer == EXIT_SEQUENCE) {
00134                exitRequested = true;
00135                break;
00136            }
00137
00138            if (inputBuffer != EXIT_SEQUENCE.substr(0, inputBuffer.length())) {
00139                uart_write_blocking(GPS_UART_PORT,
00140                    reinterpret_cast<const uint8_t*>(&ch), 1);
00141        }
00142    }
00143 }
```

```

00142         }
00143
00144     while (uart_is_readable(GPS_UART_PORT)) {
00145         char gpsByte = uart_getc(GPS_UART_PORT);
00146         uart_write_blocking(DEBUG_UART_PORT,
00147             reinterpret_cast<const uint8_t*>(&gpsByte), 1);
00148     }
00149
00150     if (to_ms_since_boot(get_absolute_time()) - startTime >= timeoutMs) {
00151         break;
00152     }
00153 }
00154
00155     uart_set_baudrate(DEBUG_UART_PORT, originalBaudRate);
00156
00157     sleep_ms(10);
00158
00159     EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_END);
00160
00161     std::string exitReason = exitRequested ? "USER_EXIT" : "TIMEOUT";
00162     std::string response = "GPS UART BRIDGE EXIT: " + exitReason;
00163     uart_print(response, VerbosityLevel::INFO);
00164
00165     frames.push_back(frame_build(OperationType::RES, 7, 2, response));
00166     return frames;
00167 }
00168
00169
00182 std::vector<Frame> handle_get_rmc_data(const std::string& param, OperationType operationType) {
00183     std::vector<Frame> frames;
00184     if (operationType != OperationType::GET) {
00185         frames.push_back(frame_build(OperationType::ERR, 7, 3, "INVALID OPERATION"));
00186         return frames;
00187     }
00188
00189     if (!param.empty()) {
00190         frames.push_back(frame_build(OperationType::ERR, 7, 3, "PARAM UNNECESSARY"));
00191         return frames;
00192     }
00193
00194     std::vector<std::string> tokens = nmea_data.get_rmc_tokens();
00195     if (tokens.empty()) {
00196         frames.push_back(frame_build(OperationType::ERR, 7, 3, "NO RMC DATA"));
00197         return frames;
00198     }
00199
00200     // Join tokens with commas to create the response
00201     std::stringstream ss;
00202     for (size_t i = 0; i < tokens.size(); ++i) {
00203         ss << tokens[i];
00204         if (i < tokens.size() - 1) {
00205             ss << ",";
00206         }
00207     }
00208
00209     frames.push_back(frame_build(OperationType::VAL, 7, 3, ss.str()));
00210     return frames;
00211 }
00212
00213
00226 std::vector<Frame> handle_get_gga_data(const std::string& param, OperationType operationType) {
00227     std::vector<Frame> frames;
00228     if (operationType != OperationType::GET) {
00229         frames.push_back(frame_build(OperationType::ERR, 7, 4, "INVALID OPERATION"));
00230         return frames;
00231     }
00232
00233     if (!param.empty()) {
00234         frames.push_back(frame_build(OperationType::ERR, 7, 4, "PARAM UNNECESSARY"));
00235         return frames;
00236     }
00237
00238     std::vector<std::string> tokens = nmea_data.get_gga_tokens();
00239     if (tokens.empty()) {
00240         frames.push_back(frame_build(OperationType::ERR, 7, 4, "NO GGA DATA"));
00241         return frames;
00242     }
00243
00244     // Join tokens with commas to create the response
00245     std::stringstream ss;
00246     for (size_t i = 0; i < tokens.size(); ++i) {
00247         ss << tokens[i];
00248         if (i < tokens.size() - 1) {
00249             ss << ",";
00250         }
00251     }
00252 }
```

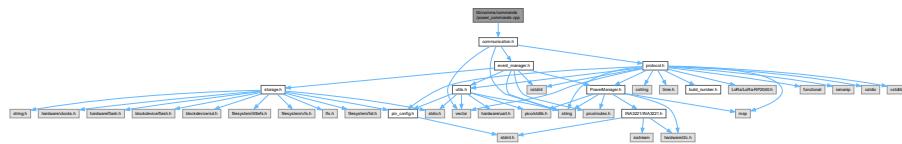
```

00253     frames.push_back(frame_build(OperationType::VAL, 7, 4, ss.str()));
00254     return frames;
00255 } // end of GPSCommands group

```

8.21 lib/comms/commands/power_commands.cpp File Reference

#include "communication.h"
Include dependency graph for power_commands.cpp:



Functions

- std::vector< Frame > handle_get_power_manager_ids (const std::string ¶m, OperationType operationType)

Handler for retrieving Power Manager IDs.
- std::vector< Frame > handle_get_voltage_battery (const std::string ¶m, OperationType operationType)

Handler for getting battery voltage.
- std::vector< Frame > handle_get_voltage_5v (const std::string ¶m, OperationType operationType)

Handler for getting 5V rail voltage.
- std::vector< Frame > handle_get_current_charge_usb (const std::string ¶m, OperationType operationType)

Handler for getting USB charge current.
- std::vector< Frame > handle_get_current_charge_solar (const std::string ¶m, OperationType operationType)

Handler for getting solar panel charge current.
- std::vector< Frame > handle_get_current_charge_total (const std::string ¶m, OperationType operationType)

Handler for getting total charge current.
- std::vector< Frame > handle_get_current_draw (const std::string ¶m, OperationType operationType)

Handler for getting system current draw.

8.22 power_commands.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002
00008
00021 std::vector<Frame> handle_get_power_manager_ids(const std::string& param, OperationType operationType)
{
00022     std::vector<Frame> frames;
00023     if (!param.empty()) {
00024         frames.push_back(frame_build(OperationType::ERR, 2, 0, "PARAM UNECESSARY"));
00025         return frames;
00026     }
00027
00028     if (!(operationType == OperationType::GET)) {
00029         frames.push_back(frame_build(OperationType::ERR, 2, 0, "INVALID OPERATION"));
00030         return frames;
00031     }

```

```

00032
00033     extern PowerManager powerManager;
00034     std::string powerManagerIDS = powerManager.read_device_ids();
00035     frames.push_back(frame_build(OperationType::VAL, 2, 0, powerManagerIDS));
00036     return frames;
00037 }
00038
00051 std::vector<Frame> handle_get_voltage_battery(const std::string& param, OperationType operationType) {
00052     std::vector<Frame> frames;
00053     if (!param.empty()) {
00054         frames.push_back(frame_build(OperationType::ERR, 2, 2, "PARAM UNECESSARY"));
00055         return frames;
00056     }
00057
00058     if (!(operationType == OperationType::GET)) {
00059         frames.push_back(frame_build(OperationType::ERR, 2, 2, "NOT ALLOWED"));
00060         return frames;
00061     }
00062
00063     extern PowerManager powerManager;
00064     float voltage = powerManager.get_voltage_battery();
00065     frames.push_back(frame_build(OperationType::VAL, 2, 2, std::to_string(voltage), ValueUnit::VOLT));
00066     return frames;
00067 }
00068
00081 std::vector<Frame> handle_get_voltage_5v(const std::string& param, OperationType operationType) {
00082     std::vector<Frame> frames;
00083     if (!param.empty()) {
00084         frames.push_back(frame_build(OperationType::ERR, 2, 3, "PARAM UNECESSARY"));
00085         return frames;
00086     }
00087
00088     if (!(operationType == OperationType::GET)) {
00089         frames.push_back(frame_build(OperationType::ERR, 2, 3, "NOT ALLOWED"));
00090         return frames;
00091     }
00092
00093     extern PowerManager powerManager;
00094     float voltage = powerManager.get_voltage_5v();
00095     frames.push_back(frame_build(OperationType::VAL, 2, 3, std::to_string(voltage), ValueUnit::VOLT));
00096     return frames;
00097 }
00098
00111 std::vector<Frame> handle_get_current_charge_usb(const std::string& param, OperationType
00112     operationType) {
00113     std::vector<Frame> frames;
00114     if (!param.empty()) {
00115         frames.push_back(frame_build(OperationType::ERR, 2, 4, "PARAM UNECESSARY"));
00116         return frames;
00117     }
00118
00119     if (!(operationType == OperationType::GET)) {
00120         frames.push_back(frame_build(OperationType::ERR, 2, 4, "NOT ALLOWED"));
00121         return frames;
00122     }
00123
00124     extern PowerManager powerManager;
00125     float chargeCurrent = powerManager.get_current_charge_usb();
00126     frames.push_back(frame_build(OperationType::VAL, 2, 4, std::to_string(chargeCurrent),
00127         ValueUnit::MILIAMP));
00128     return frames;
00129
00141 std::vector<Frame> handle_get_current_charge_solar(const std::string& param, OperationType
00142     operationType) {
00143     std::vector<Frame> frames;
00144     if (!param.empty()) {
00145         frames.push_back(frame_build(OperationType::ERR, 2, 5, "PARAM UNECESSARY"));
00146         return frames;
00147     }
00148
00149     if (!(operationType == OperationType::GET)) {
00150         frames.push_back(frame_build(OperationType::ERR, 2, 5, "NOT ALLOWED"));
00151         return frames;
00152     }
00153
00154     extern PowerManager powerManager;
00155     float chargeCurrent = powerManager.get_current_charge_solar();
00156     frames.push_back(frame_build(OperationType::VAL, 2, 5, std::to_string(chargeCurrent),
00157         ValueUnit::MILIAMP));
00158     return frames;
00159 }
00171 std::vector<Frame> handle_get_current_charge_total(const std::string& param, OperationType
00172     operationType) {
00173     std::vector<Frame> frames;
00174     if (!param.empty()) {

```

```

00174     frames.push_back(frame_build(OperationType::ERR, 2, 6, "PARAM UNECESSARY"));
00175     return frames;
00176 }
00177
00178 if (!(operationType == OperationType::GET)) {
00179     frames.push_back(frame_build(OperationType::ERR, 2, 6, "NOT ALLOWED"));
00180     return frames;
00181 }
00182
00183 extern PowerManager powerManager;
00184 float chargeCurrent = powerManager.get_current_charge_total();
00185 frames.push_back(frame_build(OperationType::VAL, 2, 6, std::to_string(chargeCurrent),
00186     ValueUnit::MILLIAMP));
00187 return frames;
00188
00189 std::vector<Frame> handle_get_current_draw(const std::string& param, OperationType operationType) {
00190     std::vector<Frame> frames;
00191     if (!param.empty()) {
00192         frames.push_back(frame_build(OperationType::ERR, 2, 7, "PARAM UNECESSARY"));
00193         return frames;
00194     }
00195
00196     if (!(operationType == OperationType::GET)) {
00197         frames.push_back(frame_build(OperationType::ERR, 2, 7, "NOT ALLOWED"));
00198         return frames;
00199     }
00200
00201     extern PowerManager powerManager;
00202     float currentDraw = powerManager.get_current_draw();
00203     frames.push_back(frame_build(OperationType::VAL, 2, 7, std::to_string(currentDraw),
00204         ValueUnit::MILLIAMP));
00205     return frames;
00206 } // end of PowerCommands group

```

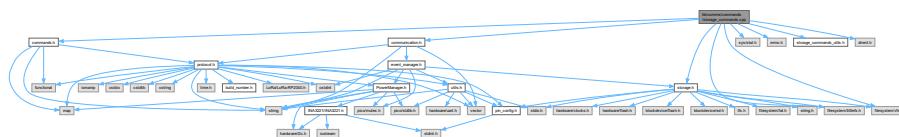
8.23 lib/comms/commands/storage_commands.cpp File Reference

```

#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 "storage_commands_utils.h"
#include "dirent.h"

Include dependency graph for storage_commands.cpp:

```



Macros

- #define MAX_BLOCK_SIZE 250
- #define STORAGE_GROUP 6
- #define START_COMMAND 1
- #define DATA_COMMAND 2
- #define END_COMMAND 3
- #define LIST_FILES_COMMAND 0
- #define MOUNT_COMMAND 4

Functions

- std::vector< Frame > handle_file_download (const std::string ¶m, OperationType operationType)
Handles the file download command.
- std::vector< Frame > handle_list_files (const std::string ¶m, OperationType operationType)
Handles the list files command.
- std::vector< Frame > handle_mount (const std::string ¶m, OperationType operationType)
Handles the SD card mount/unmount command.

8.23.1 Macro Definition Documentation

8.23.1.1 MAX_BLOCK_SIZE

```
#define MAX_BLOCK_SIZE 250
```

Definition at line 11 of file [storage_commands.cpp](#).

8.23.1.2 STORAGE_GROUP

```
#define STORAGE_GROUP 6
```

Definition at line 12 of file [storage_commands.cpp](#).

8.23.1.3 START_COMMAND

```
#define START_COMMAND 1
```

Definition at line 13 of file [storage_commands.cpp](#).

8.23.1.4 DATA_COMMAND

```
#define DATA_COMMAND 2
```

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

8.23.1.5 END_COMMAND

```
#define END_COMMAND 3
```

Definition at line 15 of file [storage_commands.cpp](#).

8.23.1.6 LIST_FILES_COMMAND

```
#define LIST_FILES_COMMAND 0
```

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

8.23.1.7 MOUNT_COMMAND

```
#define MOUNT_COMMAND 4
```

Definition at line 17 of file [storage_commands.cpp](#).

8.24 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 "storage_commands_utils.h"
00009 #include "dirent.h"
00010
00011 #define MAX_BLOCK_SIZE 250
00012 #define STORAGE_GROUP 6
00013 #define START_COMMAND 1
00014 #define DATA_COMMAND 2
00015 #define END_COMMAND 3
00016 #define LIST_FILES_COMMAND 0
00017 #define MOUNT_COMMAND 4
00023
00043 std::vector<Frame> handle_file_download(const std::string& param, OperationType operationType) {
00044     std::vector<Frame> frames;
00045     if (operationType != OperationType::GET) {
00046         frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, START_COMMAND, "Invalid
00047         operation type"));
00048     }
00049
00050     const char* filename = param.c_str();
00051     FILE* file = fopen(filename, "rb");
00052
00053     if (!file)
00054         frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, START_COMMAND, "File not
00055         found"));
00056     return frames;
00057 }
00058 // Get file size
00059 fseek(file, 0, SEEK_END);
00060 size_t fileSize = ftell(file);
00061 fseek(file, 0, SEEK_SET);
00062
00063 // Send file size to ground station
00064 frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, START_COMMAND,
00065     std::to_string(fileSize)));
00066 send_frame(frames.back());
00067 size_t block_size = MAX_BLOCK_SIZE;
00068 size_t block_count = (fileSize + block_size - 1) / block_size;
00069
00070 // Send block size and count
00071 frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, START_COMMAND,
00072     std::to_string(block_size)));
00073 send_frame(frames.back());
00073 frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, START_COMMAND,
00074     std::to_string(block_count)));
00074 send_frame(frames.back());
00075
00076 uint8_t buffer[MAX_BLOCK_SIZE];
00077 size_t bytesRead;
00078 uint32_t totalChecksum = 0;
00079 size_t blockIndex = 0;
00080
00081 while ((bytesRead = fread(buffer, 1, MAX_BLOCK_SIZE, file)) > 0) {
00082     // Send data block
00083     send_data_block(buffer, bytesRead);
00084     totalChecksum = calculate_checksum(buffer, bytesRead);
00085
00086     // Wait for ACK
00087     if (!receive_ack())
00088         fclose(file);
```

```

00089         frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, DATA_COMMAND, "ACK
00090             timeout"));
00091         return frames;
00092     }
00093     blockIndex++;
00094 }
00095 fclose(file);
00096
00097 // Send end frame with checksum
00098 std::stringstream ss;
00099 ss << std::hex << totalChecksum;
00100 frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, END_COMMAND, ss.str()));
00101 send_frame(frames.back());
00102
00103 frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, END_COMMAND, "File download
00104     complete"));
00105 return frames;
00106
00124 std::vector<Frame> handle_list_files(const std::string& param, OperationType operationType) {
00125     std::vector<Frame> frames;
00126     if (operationType != OperationType::GET) {
00127         frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND, "Invalid
00128             operation type"));
00129     }
00130
00131     DIR* dir;
00132     struct dirent* ent;
00133     int fileCount = 0; // Counter for the number of files
00134     if ((dir = opendir(".")) != NULL) {
00135         // First, count the number of files
00136         while ((ent = readdir(dir)) != NULL) {
00137             const char* filename = ent->d_name;
00138             if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0) {
00139                 continue;
00140             }
00141             fileCount++;
00142         }
00143         closedir(dir);
00144
00145         // Send the number of files
00146         frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, LIST_FILES_COMMAND,
00147             std::to_string(fileCount)));
00148
00149         // Open the directory again to read file information
00150         dir = opendir("/");
00151         if (dir != NULL) {
00152             while ((ent = readdir(dir)) != NULL) {
00153                 const char* filename = ent->d_name;
00154
00155                 // Skip "." and ".." directories
00156                 if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0) {
00157                     continue;
00158
00159                 // Get file size
00160                 char filepath[256];
00161                 snprintf(filepath, sizeof(filepath), "/%s", filename);
00162
00163                 FILE* file = fopen(filepath, "rb");
00164                 fileSize = 0;
00165
00166                 if (file != NULL) {
00167                     fseek(file, 0, SEEK_END);
00168                     fileSize = ftell(file);
00169                     fclose(file);
00170                 }
00171
00172                 // Create and send frame with filename and size
00173                 char fileInfo[512];
00174                 snprintf(fileInfo, sizeof(fileInfo), "%s:%zu", filename, fileSize);
00175                 frames.push_back(frame_build(OperationType::SEQ, STORAGE_GROUP, LIST_FILES_COMMAND,
00176                     fileInfo));
00177             }
00178             closedir(dir);
00179             frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, LIST_FILES_COMMAND,
00180                 "SEQ_DONE"));
00181         }
00182         return frames;
00183     } else {
00184         frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND, "Could
00185             not open directory for file info"));
00186         return frames;
00187     }
00188 } else {
00189     frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND, "Could not

```

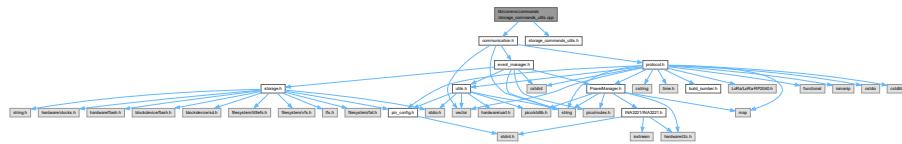
```

        open directory"));
00186     return frames;
00187 }
00188 }
00189
00206 std::vector<Frame> handle_mount(const std::string& param, OperationType operationType) {
00207     std::vector<Frame> frames;
00208     if (operationType == OperationType::GET) {
00209         frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, MOUNT_COMMAND,
00210             std::to_string(sd_card_mounted)));
00210         return frames;
00211     } else if (operationType == OperationType::SET) {
00212         if (param == "1") {
00213             if (fs_init()) {
00214                 frames.push_back(frame_build(OperationType::RES, STORAGE_GROUP, MOUNT_COMMAND, "SD
00215                     card mounted"));
00216                 return frames;
00217             } else {
00218                 frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND, "Mount
00219                     failed"));
00220             }
00221         } else if (param == "0") {
00222             if (fs_unmount("/") == 0) {
00223                 sd_card_mounted = false;
00224                 frames.push_back(frame_build(OperationType::RES, STORAGE_GROUP, MOUNT_COMMAND, "SD
00225                     card unmounted"));
00226                 return frames;
00227             } else {
00228                 frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND,
00229                     "Unmount failed"));
00230             }
00231         } else {
00232             frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND, "Invalid
00233                 parameter"));
00234         }
00235     }
00236 }
00237 } // StorageCommands

```

8.25 lib/comms/commands/storage_commands_utils.cpp File Reference

```
#include "communication.h"
#include "storage_commands_utils.h"
Include dependency graph for storage_commands_utils.cpp:
```



Functions

- uint32_t calculate_checksum (const uint8_t *data, size_t length)
- void send_data_block (const uint8_t *data, size_t length)
- bool receive_ack ()

8.25.1 Function Documentation

8.25.1.1 calculate_checksum()

```
uint32_t calculate_checksum (
    const uint8_t * data,
    size_t length)
```

Definition at line 5 of file [storage_commands_utils.cpp](#).

Here is the caller graph for this function:



8.25.1.2 send_data_block()

```
void send_data_block (
    const uint8_t * data,
    size_t length)
```

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

Here is the caller graph for this function:



8.25.1.3 receive_ack()

```
bool receive_ack ()
```

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

Here is the caller graph for this function:



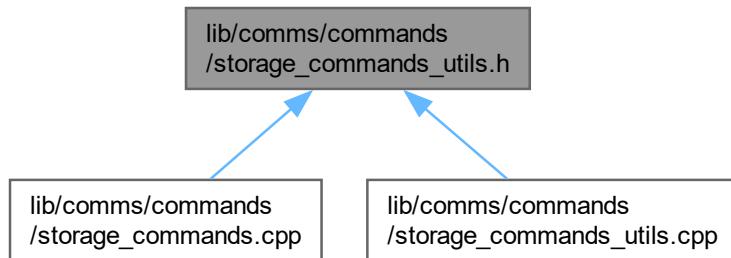
8.26 storage_commands_utils.cpp

Go to the documentation of this file.

```
00001 #include "communication.h"
00002 #include "storage_commands_utils.h"
00003
00004 // Helper function to calculate checksum (simple XOR)
00005 uint32_t calculate_checksum(const uint8_t* data, size_t length) {
00006     uint32_t checksum = 0;
00007     for (size_t i = 0; i < length; ++i) {
00008         checksum ^= data[i];
00009     }
00010     return checksum;
00011 }
00012
00013
00014 void send_data_block(const uint8_t* data, size_t length) {
00015     LoRa.beginPacket();
00016     LoRa.write(data, length);
00017     LoRa.endPacket();
00018 }
00019
00020 // Receiving an ACK (simplified)
00021 bool receive_ack() {
00022     // Implement logic to receive an ACK frame from the ground station
00023     // Return true if ACK received, false otherwise
00024     // This is a placeholder, replace with your actual ACK receiving logic
00025     return true; // Placeholder: Always return true for now
00026 }
```

8.27 lib/comms/commands/storage_commands_utils.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- `uint32_t calculate_checksum (const uint8_t *data, size_t length)`
- `void send_data_block (const uint8_t *data, size_t length)`
- `bool receive_ack ()`

8.27.1 Function Documentation

8.27.1.1 calculate_checksum()

```
uint32_t calculate_checksum (
    const uint8_t * data,
    size_t length)
```

Definition at line 5 of file [storage_commands_utils.cpp](#).

Here is the caller graph for this function:



8.27.1.2 send_data_block()

```
void send_data_block (
    const uint8_t * data,
    size_t length)
```

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

Here is the caller graph for this function:



8.27.1.3 receive_ack()

```
bool receive_ack ()
```

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

Here is the caller graph for this function:



8.28 storage_commands_utils.h

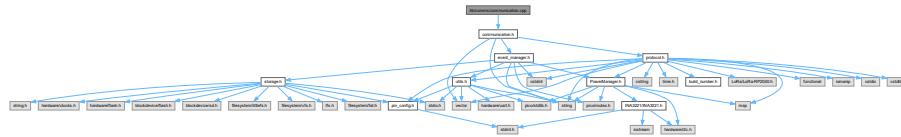
[Go to the documentation of this file.](#)

```
00001 uint32_t calculate_checksum(const uint8_t* data, size_t length);
00002 void send_data_block(const uint8_t* data, size_t length);
00003 bool receive_ack();
```

8.29 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`

8.29.1 Function Documentation

8.29.1.1 initialize_radio()

```
bool initialize_radio ()
```

Initializes the LoRa radio module.

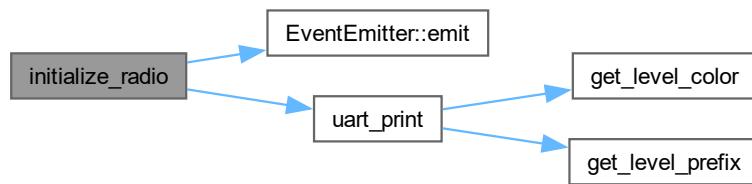
Returns

True if initialization was successful, false otherwise.

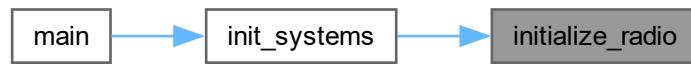
Sets the LoRa pins and attempts to begin LoRa communication at a specified frequency. Emits a [CommsEvent::RADIO_INIT](#) event on success or a [CommsEvent::RADIO_ERROR](#) event on failure.

Definition at line 17 of file [communication.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2 Variable Documentation

8.29.2.1 outgoing

```
string outgoing
```

Definition at line 3 of file [communication.cpp](#).

8.29.2.2 msgCount

```
uint8_t msgCount = 0
```

Definition at line 4 of file [communication.cpp](#).

8.29.2.3 lastSendTime

```
long lastSendTime = 0
```

Definition at line 5 of file [communication.cpp](#).

8.29.2.4 lastReceiveTime

```
long lastReceiveTime = 0
```

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

8.29.2.5 lastPrintTime

```
long lastPrintTime = 0
```

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

8.29.2.6 interval

```
unsigned long interval = 0
```

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

8.30 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 initStatus = false;
00015     if (!LoRa.begin(frequency))
00016     {
00017         uart_print("LoRa init failed. Check your connections.", VerbosityLevel::WARNING);
00018         initStatus = false;
00019     } else {
00020         uart_print("LoRa initialized with frequency " + std::to_string(frequency),
00021             VerbosityLevel::INFO);
00022         initStatus = true;
00023     }
00024     EventEmitter::emit(EventGroup::COMMS, initStatus ? CommsEvent::RADIO_INIT :
00025         CommsEvent::RADIO_ERROR);
00026     return initStatus;
00027 }
00028
00029
00030
00031
00032
00033 }
```

8.31 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 (const Frame &frame)`
- `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 ¶m, 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)`

8.31.1 Function Documentation

8.31.1.1 initialize_radio()

```
bool initialize_radio ()
```

Initializes the LoRa radio module.

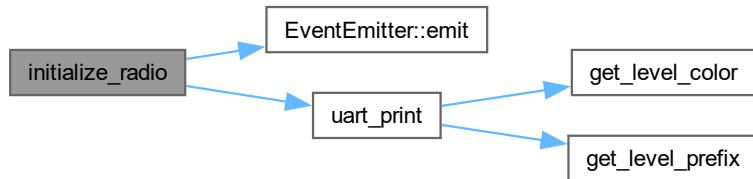
Returns

True if initialization was successful, false otherwise.

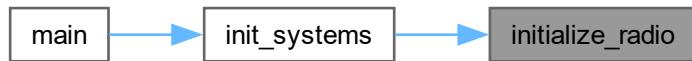
Sets the LoRa pins and attempts to begin LoRa communication at a specified frequency. Emits a [CommsEvent::RADIO_INIT](#) event on success or a [CommsEvent::RADIO_ERROR](#) event on failure.

Definition at line 17 of file [communication.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.31.1.2 on_receive()

```
void on_receive (
    int packetSize)
```

Callback function for handling received LoRa packets.

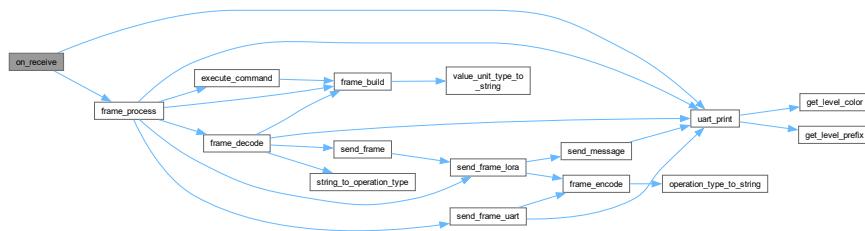
Parameters

<code>packetSize</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:



8.31.1.3 handle_uart_input()

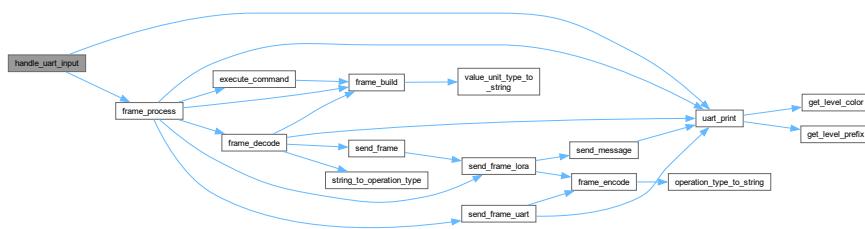
```
void handle_uart_input ()
```

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:



8.31.1.4 send_message()

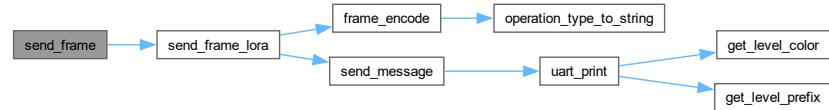
```
void send_message (
    std::string outgoing)
```

8.31.1.5 send_frame()

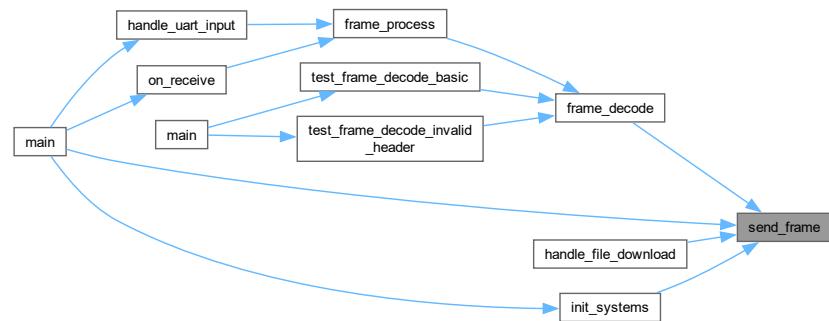
```
void send_frame (
    const Frame & frame)
```

Definition at line 48 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

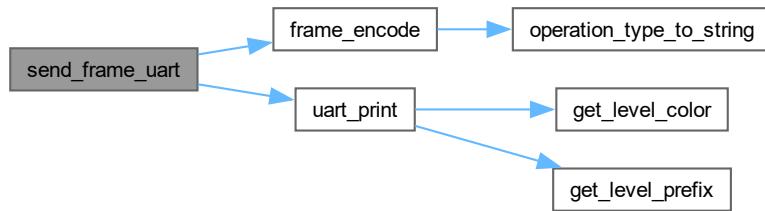


8.31.1.6 send_frame_uart()

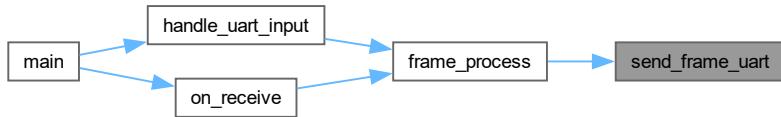
```
void send_frame_uart (
    const Frame & frame)
```

Definition at line 42 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

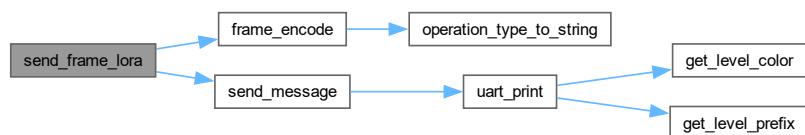


8.31.1.7 send_frame_lora()

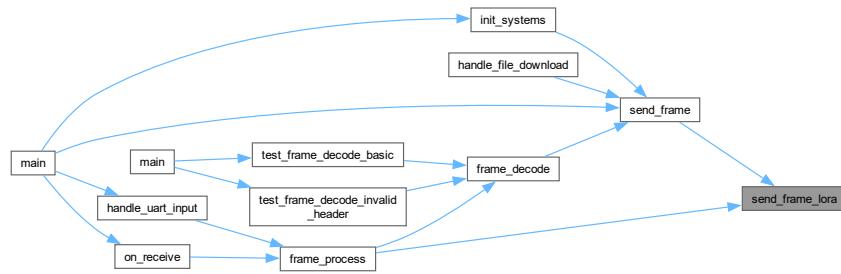
```
void send_frame_lora (
    const Frame & frame)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.31.1.8 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 59 of file [send.cpp](#).

8.31.1.9 determine_unit()

```
std::string determine_unit (
    uint8_t group,
    uint8_t command)
```

8.32 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(const Frame& frame);
```

```

00014 void send_frame_uart(const Frame& frame);
00015 void send_frame_lora(const Frame& frame);
00016
00017 void split_and_send_message(const uint8_t* data, size_t length);
00018
00019 std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType
operationType);
00020
00021 void frame_process(const std::string& data, Interface interface);
00022 std::string frame_encode(const Frame& frame);
00023 Frame frame_decode(const std::string& data);
00024 Frame frame_build(OperationType operation, uint8_t group, uint8_t command, const std::string& value,
const ValueUnit unitType = ValueUnit::UNDEFINED);
00025
00026 std::string determine_unit(uint8_t group, uint8_t command);
00027
00028 #endif

```

8.33 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 > `commandHandlers`
Global map of all command handlers.
- volatile uint16_t `eventRegister`

8.33.1 Detailed Description

Implements functions for encoding, decoding, building, and processing Frames.

Definition in file [frame.cpp](#).

8.33.2 Typedef Documentation

8.33.2.1 CommandHandler

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

Definition at line 3 of file [frame.cpp](#).

8.33.3 Variable Documentation

8.33.3.1 eventRegister

```
volatile uint16_t eventRegister [extern]
```

8.34 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> commandHandlers;
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 decodedFrameData;
00074         while (std::getline(ss, token, DELIMITER)) {
00075             if (token == FRAME_END) break;
00076             decodedFrameData += token + DELIMITER;
00077         }
00078         if (!decodedFrameData.empty())
00079             decodedFrameData.pop_back();
```

```

00080         }
00081
00082         std::stringstream frameDataStream(decodedFrameData);
00083
00084         std::getline(frameDataStream, token, DELIMITER);
00085         frame.direction = std::stoi(token);
00086
00087         std::getline(frameDataStream, token, DELIMITER);
00088         frame.operationType = string_to_operation_type(token);
00089
00090         std::getline(frameDataStream, token, DELIMITER);
00091         frame.group = std::stoi(token);
00092
00093         std::getline(frameDataStream, token, DELIMITER);
00094         frame.command = std::stoi(token);
00095
00096         std::getline(frameDataStream, token, DELIMITER);
00097         frame.value = token;
00098
00099         std::getline(frameDataStream, 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 errorFrame = frame_build(OperationType::ERR, 0, 0, e.what());
00106         send_frame(errorFrame);
00107         throw;
00108     }
00109 }
00110
00111
00118 void frame_process(const std::string& data, Interface interface) {
00119     uart_print("Processing frame: " + data, VerbosityLevel::WARNING);
00120     try {
00121         Frame frame = frame_decode(data);
00122         uint32_t commandKey = (static_cast<uint32_t>(frame.group) << 8) |
00123             static_cast<uint32_t>(frame.command);
00124
00125         std::vector<Frame> responseFrames = execute_command(commandKey, frame.value,
00126             frame.operationType);
00127
00128         // Send all responses through the same interface that received the command
00129         for (const auto& responseFrame : responseFrames) {
00130             if (interface == Interface::UART) {
00131                 send_frame_uart(responseFrame);
00132             } else if (interface == Interface::LORA) {
00133                 send_frame_lora(responseFrame);
00134                 sleep_ms(50);
00135             }
00136         } catch (const std::exception& e) {
00137             Frame errorFrame = frame_build(OperationType::ERR, 0, 0, e.what());
00138             if (interface == Interface::UART) {
00139                 send_frame_uart(errorFrame);
00140             } else if (interface == Interface::LORA) {
00141                 send_frame_lora(errorFrame);
00142             }
00143     }
00144
00155 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             frame.operationType = OperationType::RES;
00179             frame.value = value;
00180             frame.unit = value_unit_type_to_string(unitType);

```

```

00181         break;
00182
00183     case OperationType::SEQ:
00184         frame.direction = 1;
00185         frame.operationType = OperationType::SEQ;
00186         frame.value = value;
00187         frame.unit = value_unit_type_to_string(unitType);
00188         break;
00189     }
00190
00191     frame.group = group;
00192     frame.command = command;
00193
00194     return frame;
00195 }
00196 // end of FrameHandling group

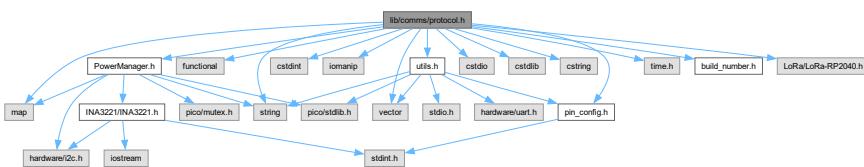
```

8.35 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 `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 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 = ','`

8.35.1 Enumeration Type Documentation

8.35.1.1 OperationType

`enum class OperationType [strong]`

Represents the type of operation being performed.

Enumerator

<code>GET</code>	Get data.
<code>SET</code>	Set data.
<code>RES</code>	Set command result.
<code>VAL</code>	Get command value.
<code>SEQ</code>	Sequence element response.
<code>ERR</code>	Error occurred during command execution.

Definition at line 45 of file `protocol.h`.

8.35.1.2 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 67 of file [protocol.h](#).

8.35.1.3 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 84 of file [protocol.h](#).

8.35.1.4 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 107 of file [protocol.h](#).

8.35.1.5 Interface

```
enum class Interface [strong]
```

Represents the communication interface being used.

Enumerator

UART	UART interface.
LORA	LoRa interface.

Definition at line 126 of file [protocol.h](#).

8.35.2 Function Documentation

8.35.2.1 exception_type_to_string()

```
std::string exception_type_to_string (
    ExceptionType type)
```

Converts an [ExceptionType](#) to a string.

Parameters

<i>type</i>	The ExceptionType 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:



8.35.2.2 operation_type_to_string()

```
std::string operation_type_to_string (
    OperationType type)
```

Converts an [OperationType](#) to a string.

Parameters

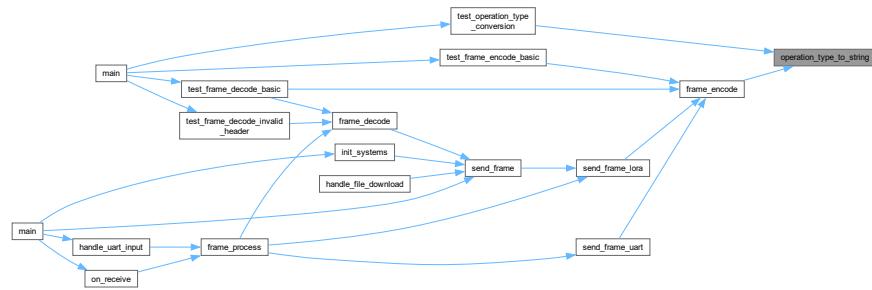
<i>type</i>	The OperationType to convert.
-------------	---

Returns

The string representation of the [OperationType](#).

Definition at line 50 of file [utils_converters.cpp](#).

Here is the caller graph for this function:

**8.35.2.3 string_to_operation_type()**

```
OperationType string_to_operation_type (
    const std::string & str)
```

Converts a string to an [OperationType](#).

Parameters

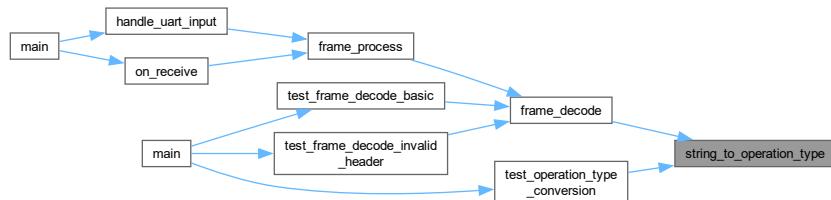
<code>str</code>	The string to convert.
------------------	------------------------

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:

**8.35.2.4 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

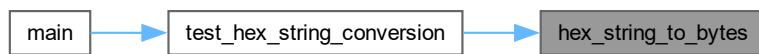
<i>hexString</i>	The hex string to convert.
------------------	----------------------------

Returns

A vector of bytes representing the hex string.

Definition at line 83 of file [utils_converters.cpp](#).

Here is the caller graph for this function:



8.35.2.5 value_unit_type_to_string()

```
std::string value_unit_type_to_string (
```

`ValueUnit unit`)

Converts a [ValueUnit](#) to a string.

Parameters

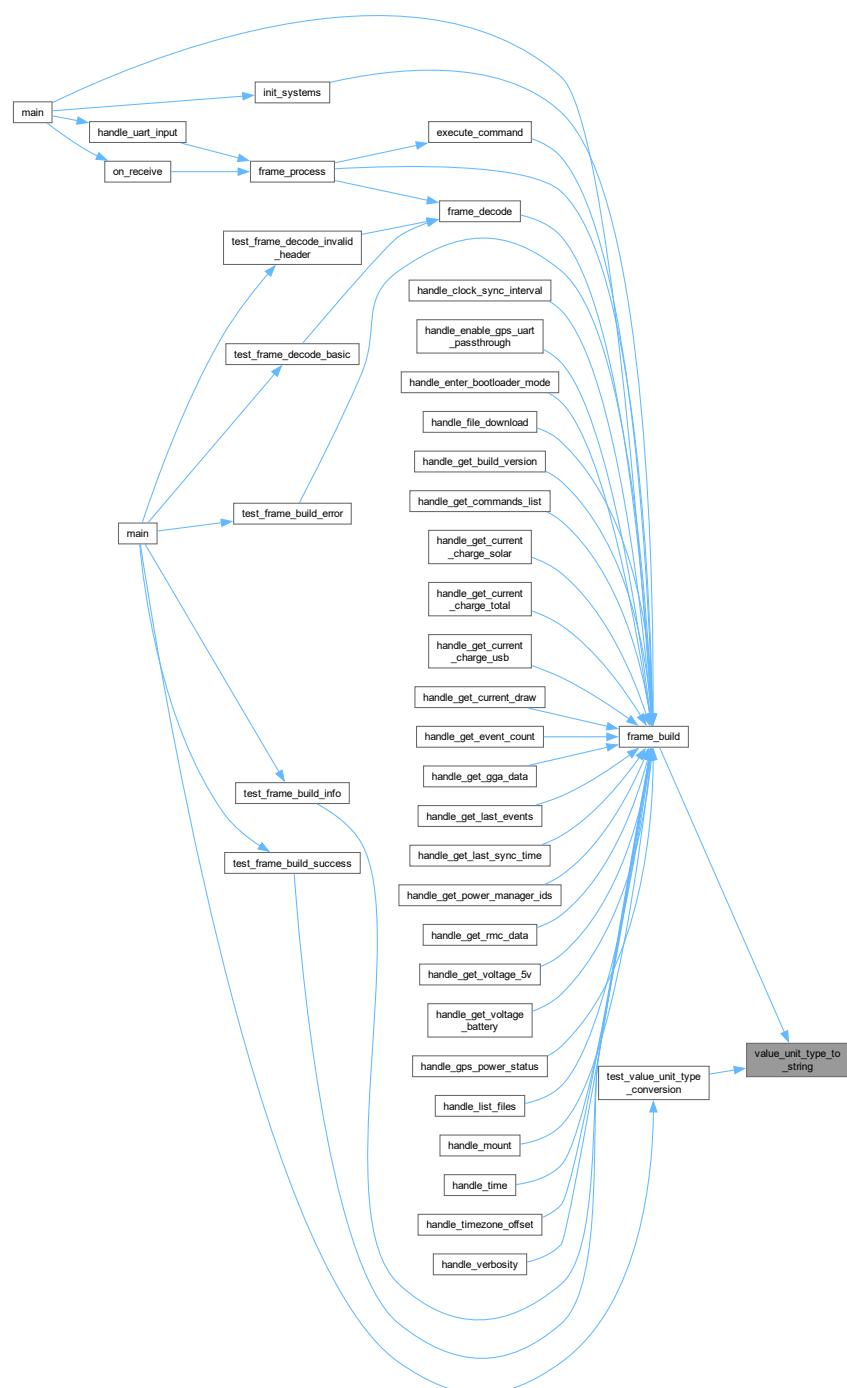
<i>unit</i>	The ValueUnit to convert.
-------------	---

Returns

The string representation of the [ValueUnit](#).

Definition at line 31 of file [utils_converters.cpp](#).

Here is the caller graph for this function:



8.35.3 Variable Documentation

8.35.3.1 FRAME_BEGIN

```
const std::string FRAME_BEGIN = "KBST"
```

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

8.35.3.2 FRAME_END

```
const std::string FRAME_END = "TSBK"
```

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

8.35.3.3 DELIMITER

```
const char DELIMITER = ';'
```

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

8.36 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 OperationType {
00047     GET,
00049     SET,
00051     RES,
00053     VAL,
00055     SEQ,
00057     ERR,
00058
00059 };
00060
00061
00062
00067 enum class CommandAccessLevel {
00069     NONE,
00071     READ_ONLY,
00073     WRITE_ONLY,
00075     READ_WRITE
00076 };
00077
00078
00079
00084 enum class ValueUnit {
00086     UNDEFINED,
00088     SECOND,
00090     VOLT,
00092     BOOL,
00094     DATETIME,
00096     TEXT,
00098     MILIAMP,
```

```

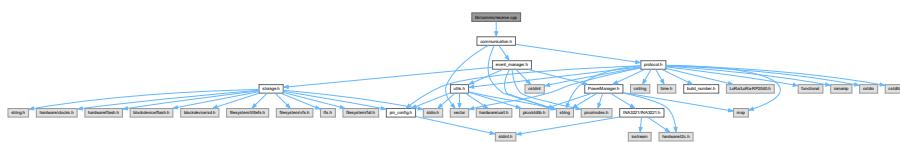
00099 };
00100
00101
00102
00107 enum class ExceptionType {
00109     NONE,
00111     NOT_ALLOWED,
00113     INVALID_PARAM,
00115     INVALID_OPERATION,
00117     PARAM_UNNECESSARY
00118 };
00119
00120
00121
00126 enum class Interface {
00128     UART,
00130     LORA
00131 };
00132
00133
00209 struct Frame {
00210     std::string header;           // Start marker
00211     uint8_t direction;          // 0 = ground->sat, 1 = sat->ground
00212     OperationType operationType;
00213     uint8_t group;               // Group ID
00214     uint8_t command;             // Command ID within group
00215     std::string value;            // Payload value
00216     std::string unit;             // Payload unit
00217     std::string footer;           // End marker
00218 };
00219
00220 std::string exception_type_to_string(ExceptionType type);
00221 std::string operation_type_to_string(OperationType type);
00222 OperationType string_to_operation_type(const std::string& str);
00223 std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString);
00224 std::string value_unit_type_to_string(ValueUnit unit);
00225
00226 #endif

```

8.37 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 packetSize)
Callback function for handling received LoRa packets.
- void `handle_uart_input` ()
Handles UART input.

8.37.1 Detailed Description

Implements functions for receiving and processing data, including LoRa and UART input.

Definition in file [receive.cpp](#).

8.37.2 Function Documentation

8.37.2.1 on_receive()

```
void on_receive (
    int packetSize)
```

Callback function for handling received LoRa packets.

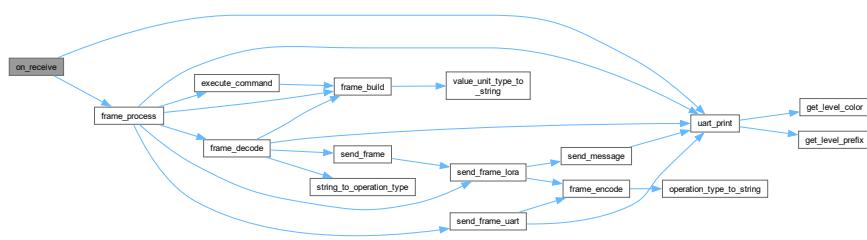
Parameters

<i>packetSize</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:



8.37.2.2 handle_uart_input()

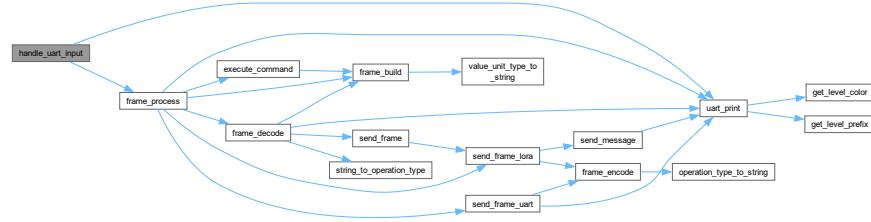
```
void handle_uart_input ()
```

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:



8.38 receive.cpp

[Go to the documentation of this file.](#)

```

00001 #include "communication.h"
00002
00003
00008
00015 void on_receive(int packetSize) {
00016     if (packetSize == 0) return;
00017
00018     uint8_t buffer[256];
00019     int bytesRead = 0;
00020
00021     while (LoRa.available() && bytesRead < packetSize) {
00022         buffer[bytesRead++] = LoRa.read();
00023     }
00024
00025     // Extract LoRa metadata
00026     uint8_t receivedDestination = buffer[0];
00027     uint8_t receivedLocalAddress = buffer[1];
00028
00029     // Validate metadata (optional, for security)
00030     if (receivedDestination != lora_address_local) {
00031         uart_print("Error: Destination address mismatch!", VerbosityLevel::ERROR);
00032         return;
00033     }
00034
00035     if (receivedLocalAddress != 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 startIndex = 2; // Start after the metadata
00042
00043     // Extract the frame data
00044     std::string received = std::string(reinterpret_cast<char*>(buffer + startIndex), bytesRead -
00045     startIndex);
  
```

```

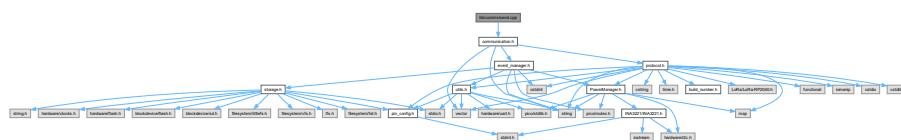
00046     if (received.empty()) return;
00047
00048     // Debug: Print raw hex values
00049     std::stringstream hexDump;
00050     hexDump << "Raw bytes: ";
00051     for (int i = 0; i < bytesRead; i++) {
00052         hexDump << std::hex << std::setfill('0') << std::setw(2)
00053             << static_cast<int>(buffer[i]) << " ";
00054     }
00055     uart_print(hexDump.str(), VerbosityLevel::DEBUG);
00056
00057     // Find frame boundaries
00058     size_t headerPos = received.find(FRAME_BEGIN);
00059     size_t footerPos = received.find(FRAME_END);
00060
00061     if (headerPos != std::string::npos && footerPos != std::string::npos && footerPos > headerPos) {
00062         std::string frameData = received.substr(headerPos, footerPos + FRAME_END.length() -
00063             headerPos);
00063         uart_print("Extracted frame (length=" + std::to_string(frameData.length()) + "): " +
00064             frameData, VerbosityLevel::DEBUG);
00065     } else {
00066         uart_print("No valid frame found in received data", VerbosityLevel::WARNING);
00067     }
00068 }
00069
00070
00071 void handle_uart_input() {
00072     static std::string uartBuffer;
00073
00074     while (uart_is_readable(DEBUG_UART_PORT)) {
00075         char c = uart_getc(DEBUG_UART_PORT);
00076
00077         if (c == '\r' || c == '\n') {
00078             uart_print("Received UART string: " + uartBuffer, VerbosityLevel::DEBUG);
00079             frame_process(uartBuffer, Interface::UART);
00080             uartBuffer.clear();
00081         } else {
00082             uartBuffer += c;
00083         }
00084     }
00085 }
00086
00087 }
00088
00089
00090 }
```

8.39 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 send_frame (const Frame &frame)`
- `void split_and_send_message (const uint8_t *data, size_t length)`
Sends a large packet using LoRa.

8.39.1 Detailed Description

Implements functions for sending data, including LoRa messages and Frames.

Definition in file [send.cpp](#).

8.39.2 Function Documentation

8.39.2.1 send_message()

```
void send_message (
    string outgoing)
```

Sends a message using LoRa.

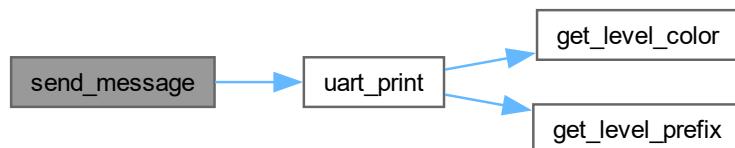
Parameters

<i>outgoing</i>	The message to send.
-----------------	----------------------

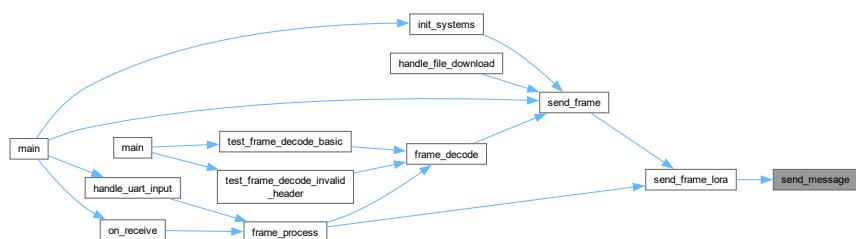
Converts the outgoing string to a C-style string, adds destination and local addresses, and sends the message using LoRa. Prints a log message to the UART.

Definition at line 15 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

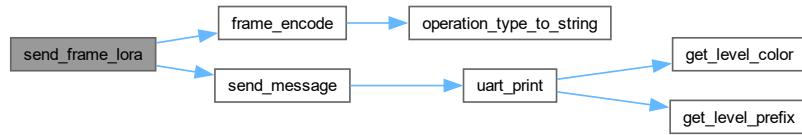


8.39.2.2 send_frame_lora()

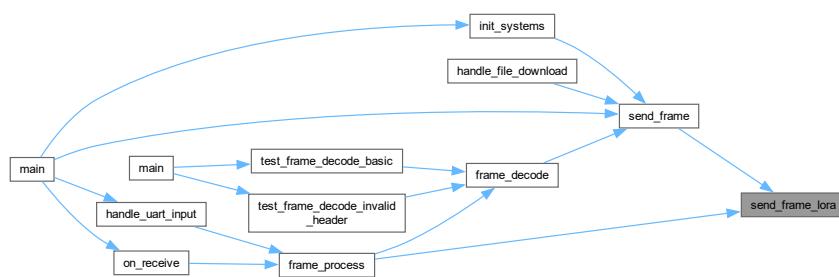
```
void send_frame_lora (
    const Frame & frame)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

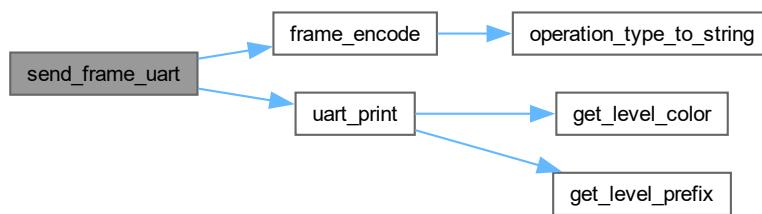


8.39.2.3 send_frame_uart()

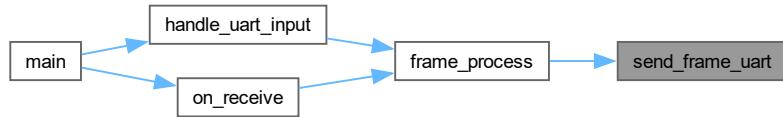
```
void send_frame_uart (
    const Frame & frame)
```

Definition at line 42 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

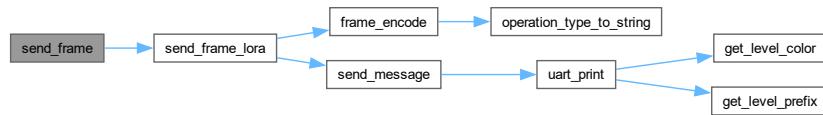


8.39.2.4 send_frame()

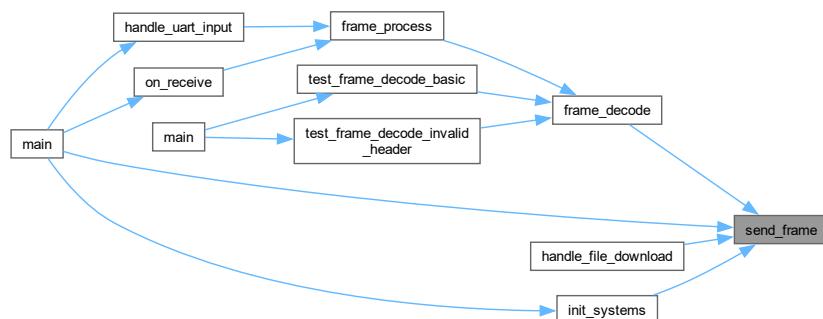
```
void send_frame (
    const Frame & frame)
```

Definition at line 48 of file [send.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.39.2.5 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 59 of file [send.cpp](#).

8.40 send.cpp

[Go to the documentation of this file.](#)

```

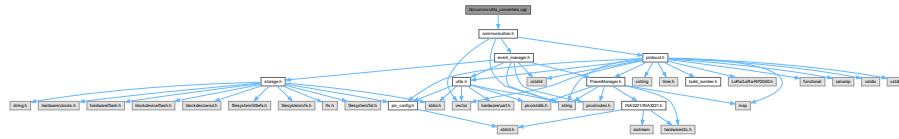
00001 #include "communication.h"
00002
00003
00008
00015 void send_message(string outgoing)
00016 {
00017     int n = outgoing.length();
00018     char send[n + 1];
00019     strcpy(send, outgoing.c_str());
00020
00021     LoRa.beginPacket();           // start packet
00022     LoRa.write(lora_address_remote); // 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 messageToLog = "Sent message of size " + std::to_string(n);
00028     messageToLog += " to 0x" + std::to_string(lora_address_remote);
00029     messageToLog += " containing: " + string(send);
00030
00031     uart_print(messageToLog, VerbosityLevel::DEBUG);
00032
00033     LoRa.flush();
00034 }
00035
00036
00037 void send_frame_lora(const Frame& frame) {
00038     std::string encodedFrame = frame_encode(frame);
00039     send_message(encodedFrame);
00040 }
00041
00042 void send_frame_uart(const Frame& frame) {
00043     std::string encodedFrame = frame_encode(frame);
00044     uart_print(encodedFrame);
00045 }
00046
00047 [[deprecated("Use send_frame_lora or send_frame_uart instead")]]
00048 void send_frame(const Frame& frame) {
00049     send_frame_lora(frame);
00050 }
00051
00052
00059 void split_and_send_message(const uint8_t* data, size_t length)
00060 {
00061     const size_t MAX_PKT_SIZE = 255;
00062     size_t offset = 0;
00063     while (offset < length)
00064     {
00065         size_t chunkSize = ((length - offset) < MAX_PKT_SIZE) ? (length - offset) : MAX_PKT_SIZE;
00066         LoRa.beginPacket();
00067         LoRa.write(&data[offset], chunkSize);
00068         LoRa.endPacket();
00069         offset += chunkSize;
00070         sleep_ms(100);
00071     }
00072 }
00073

```

8.41 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::vector< uint8_t > [hex_string_to_bytes](#) (const std::string &hexString)
Converts a hex string to a vector of bytes.

8.41.1 Detailed Description

Implements utility functions for converting between different data types.

Definition in file [utils_converters.cpp](#).

8.41.2 Function Documentation

8.41.2.1 exception_type_to_string()

```
std::string exception_type_to_string (
    ExceptionType type)
```

Converts an [ExceptionType](#) to a string.

Parameters

<i>type</i>	The ExceptionType 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:



8.41.2.2 `value_unit_type_to_string()`

```
std::string value_unit_type_to_string (
    ValueUnit unit)
```

Converts a [ValueUnit](#) to a string.

Parameters

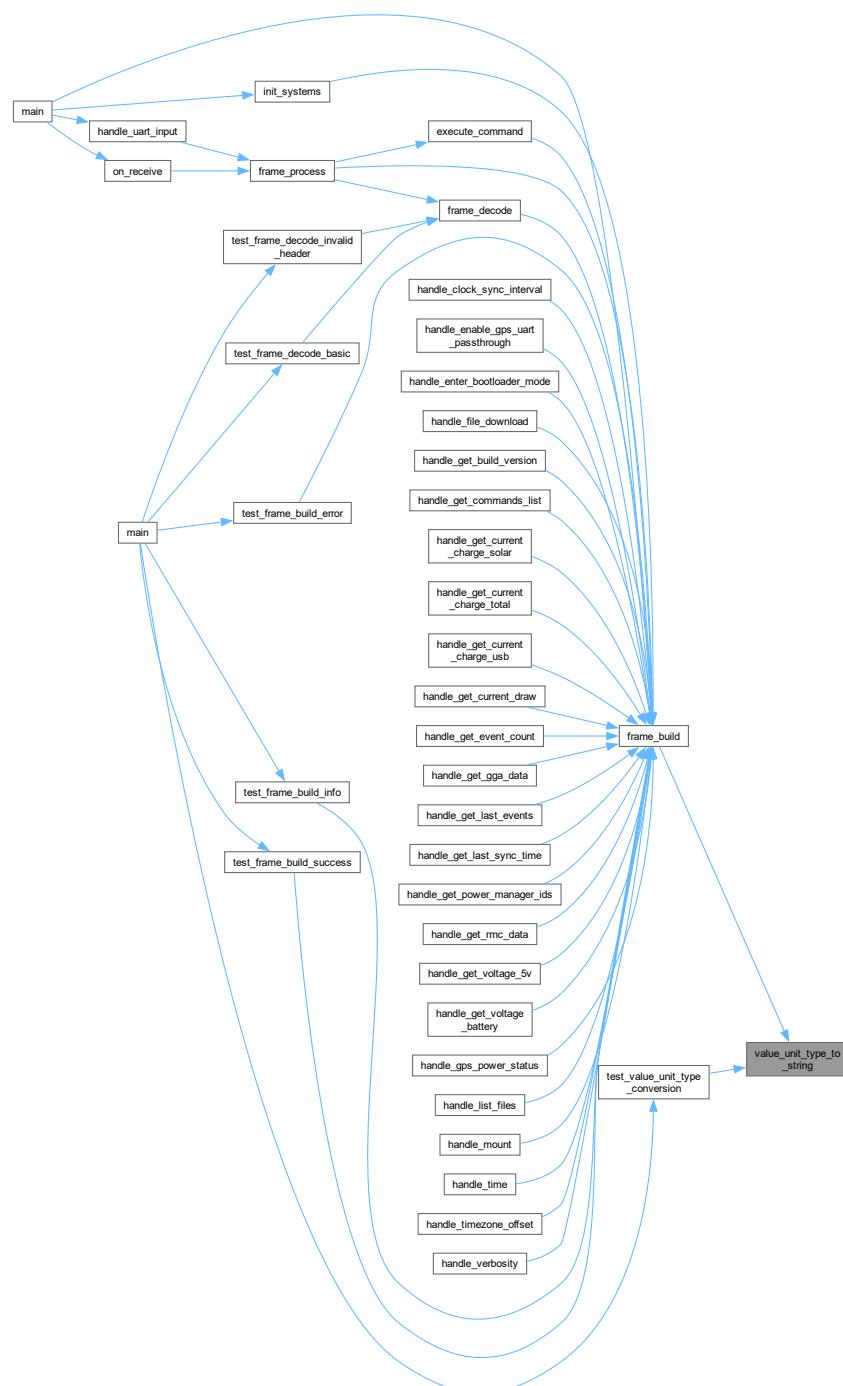
<i>unit</i>	The ValueUnit to convert.
-------------	---

Returns

The string representation of the [ValueUnit](#).

Definition at line 31 of file [utils_converters.cpp](#).

Here is the caller graph for this function:



8.41.2.3 operation_type_to_string()

```
std::string operation_type_to_string (
    OperationType type)
```

Converts an `OperationType` to a string.

Parameters

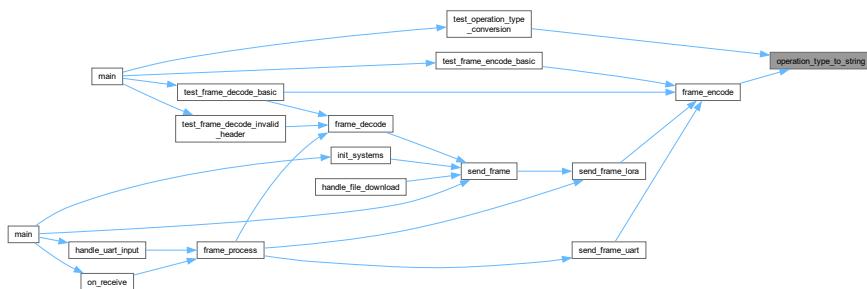
<code>type</code>	The OperationType to convert.
-------------------	---

Returns

The string representation of the [OperationType](#).

Definition at line 50 of file [utils_converters.cpp](#).

Here is the caller graph for this function:

**8.41.2.4 string_to_operation_type()**

```
OperationType string_to_operation_type (
    const std::string & str)
```

Converts a string to an [OperationType](#).

Parameters

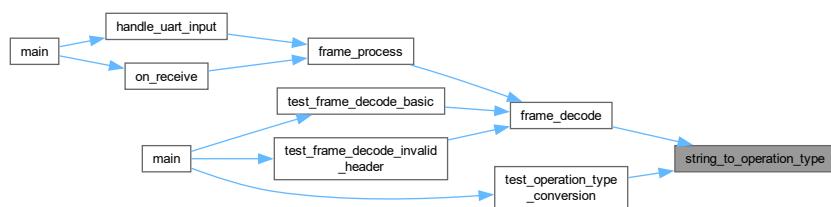
<code>str</code>	The string to convert.
------------------	------------------------

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:



8.41.2.5 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

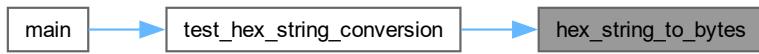
<i>hexString</i>	The hex string to convert.
------------------	----------------------------

Returns

A vector of bytes representing the hex string.

Definition at line 83 of file [utils_converters.cpp](#).

Here is the caller graph for this function:



8.42 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
00063 OperationType string_to_operation_type(const std::string& str) {
00064     if (str == "GET") return OperationType::GET;
00065     if (str == "SET") return OperationType::SET;
00066     if (str == "VAL") return OperationType::VAL;
00067     if (str == "ERR") return OperationType::ERR;
00068     if (str == "RES") return OperationType::RES;
00069     if (str == "SEQ") return OperationType::SEQ;
00070     return OperationType::GET; // Default to GET
00071 }
00072
00073 std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString) {
00074     std::vector<uint8_t> bytes;
00075     for (size_t i = 0; i < hexString.length(); i += 2) {
00076         std::string byteString = hexString.substr(i, 2);
00077         unsigned int byte;
00078         std::stringstream ss;
00079         ss << std::hex << byteString;
00080         ss >> byte;
00081         bytes.push_back(static_cast<uint8_t>(byte));
00082     }
00083     return bytes;
00084 }
00085

```

8.43 lib/eventman/event_manager.cpp File Reference

Implements the event management system for the Kabisat 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:
```



Functions

- void [check_power_events \(PowerManager &pm\)](#)
Checks power statuses and triggers events based on voltage trends.

Variables

- volatile uint16_t [eventLogId = 0](#)
Global event log ID counter.
- static PowerEvent [lastPowerState = PowerEvent::LOW_BATTERY](#)

- static constexpr float **FALL_RATE_THRESHOLD** = -0.02f
Threshold for detecting a falling voltage rate.
- static constexpr int **FALLING_TREND_REQUIRED** = 3
Number of consecutive falling voltage readings required to trigger a power falling event.
- static constexpr float **VOLTAGE_LOW_THRESHOLD** = 4.7f
Voltage threshold for detecting a low battery condition.
- static constexpr float **VOLTAGE_OVERCHARGE_THRESHOLD** = 5.3f
Voltage threshold for detecting an overcharge condition.
- static int **fallingTrendCount** = 0
Counter for consecutive falling voltage readings.
- bool **lastSolarState** = false
Stores the last known solar charging state.
- bool **lastUSBState** = false
Stores the last known USB connection state.
- DS3231 **systemClock**
External declaration of the system clock.
- EventManagerImpl **eventManager**
Global instance of the [EventManager](#) implementation.

8.43.1 Detailed Description

Implements the event management system for the Kubesat 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](#).

8.44 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
00017
00022 volatile uint16_t eventLogId = 0;
00023
00028 static PowerEvent lastPowerState = PowerEvent::LOW_BATTERY;
00029
00034 static constexpr float FALL_RATE_THRESHOLD = -0.02f;
00035
00040 static constexpr int FALLING_TREND_REQUIRED = 3;
00041
00046 static constexpr float VOLTAGE_LOW_THRESHOLD = 4.7f;
00047
00052 static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f;
00053
00058 static int fallingTrendCount = 0;
00059
00064 bool lastSolarState = false;
00065
00070 bool lastUSBState = false;
00071

```

```

00076 extern DS3231 systemClock;
00077
00082 EventManagerImpl eventManager;
00083
00084
00093 void EventManager::log_event(uint8_t group, uint8_t event) {
00094     mutex_enter_blocking(&eventMutex);
00095
00096     EventLog& log = events[writeIndex];
00097     log.id = nextEventId++;
00098     log.timestamp = systemClock.get_unix_time();
00099     log.group = group;
00100     log.event = event;
00101
00102     // Print event immediately
00103     uart_print(log.to_string(), VerbosityLevel::EVENT);
00104
00105     writeIndex = (writeIndex + 1) % EVENT_BUFFER_SIZE;
00106     if (eventCount < EVENT_BUFFER_SIZE) {
00107         eventCount++;
00108     }
00109
00110     // Set persistence flag on buffer full or power events
00111     if (eventCount == EVENT_BUFFER_SIZE ||
00112         (group == static_cast<uint8_t>(EventGroup::POWER) &&
00113          event == static_cast<uint8_t>(PowerEvent::POWER_FALLING))) {
00114         needsPersistence = true;
00115         save_to_storage();
00116     }
00117
00118     mutex_exit(&eventMutex);
00119 }
00120
00121
00128 const EventLog& EventManager::get_event(size_t index) const {
00129     static const EventLog emptyEvent = {0, 0, 0, 0}; // Initialize {id, timestamp, group, event}
00130     if (index >= eventCount) {
00131         return emptyEvent;
00132     }
00133
00134     // Calculate actual index in circular buffer
00135     size_t actualIndex;
00136     if (eventCount == EVENT_BUFFER_SIZE) {
00137         actualIndex = (writeIndex + index) % EVENT_BUFFER_SIZE;
00138     } else {
00139         actualIndex = index;
00140     }
00141
00142     return events[actualIndex];
00143 }
00144
00145
00154 void check_power_events(PowerManager& pm) {
00155     float currentVoltage = pm.get_voltage_5v();
00156     static float previousVoltage = 0.0f;
00157     float delta = currentVoltage - previousVoltage;
00158     previousVoltage = currentVoltage;
00159
00160     if (delta < FALL_RATE_THRESHOLD) {
00161         fallingTrendCount++;
00162     } else {
00163         fallingTrendCount = 0;
00164     }
00165
00166     if (fallingTrendCount >= FALLING_TREND_REQUIRED) {
00167         lastPowerState = PowerEvent::POWER_FALLING;
00168         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_FALLING);
00169     }
00170
00171     if (currentVoltage < PowerManager::VOLTAGE_LOW_THRESHOLD &&
00172         lastPowerState != PowerEvent::LOW_BATTERY) {
00173         lastPowerState = PowerEvent::LOW_BATTERY;
00174         EventEmitter::emit(EventGroup::POWER, PowerEvent::LOW_BATTERY);
00175     }
00176     else if (currentVoltage > PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&
00177               lastPowerState != PowerEvent::OVERCHARGE) {
00178         lastPowerState = PowerEvent::OVERCHARGE;
00179         EventEmitter::emit(EventGroup::POWER, PowerEvent::OVERCHARGE);
00180     }
00181     else if (currentVoltage >= PowerManager::VOLTAGE_LOW_THRESHOLD &&
00182               currentVoltage <= PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&
00183               lastPowerState != PowerEvent::POWER_NORMAL) {
00184         lastPowerState = PowerEvent::POWER_NORMAL;
00185         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_NORMAL);
00186     }
00187
00188     // Check solar charging state

```

```

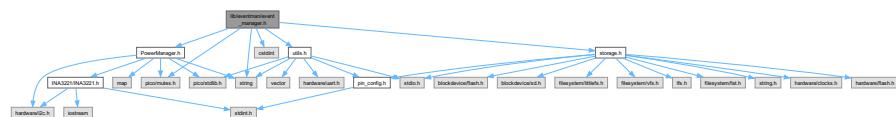
00189     bool currentSolarState = pm.is_charging_solar();
00190     if (currentSolarState != lastSolarState) {
00191         if (currentSolarState) {
00192             EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_ACTIVE);
00193         } else {
00194             EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_INACTIVE);
00195         }
00196         lastSolarState = currentSolarState;
00197     }
00198
00199     // Check USB connection state
00200     bool currentUSBState = pm.is_charging_usb();
00201     if (currentUSBState != lastUSBState) {
00202         if (currentUSBState) {
00203             EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_CONNECTED);
00204         } else {
00205             EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_DISCONNECTED);
00206         }
00207         lastUSBState = currentUSBState;
00208     }
00209 }
```

8.45 lib/eventman/event_manager.h File Reference

Manages the event logging system for the Kubesat 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 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 }`

Represents specific clock-related events.

Functions

- class `EventLog __attribute__ ((packed))`
- `std::string to_string () const`

Converts the `EventLog` to a string representation.
- `void check_power_events (PowerManager &pm)`

Checks power statuses and triggers events based on voltage trends.

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.

8.45.1 Detailed Description

Manages the event logging system for the Kabisat firmware.

Definition in file [event_manager.h](#).

8.45.2 Macro Definition Documentation

8.45.2.1 EVENT_BUFFER_SIZE

```
#define EVENT_BUFFER_SIZE 10
```

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

8.45.2.2 EVENT_LOG_FILE

```
#define EVENT_LOG_FILE "/event_log.csv"
```

Definition at line 12 of file [event_manager.h](#).

8.45.3 Function Documentation

8.45.3.1 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](#).

8.45.4 Variable Documentation

8.45.4.1 id

```
uint16_t id
```

Sequence number.

Definition at line [2](#) of file [event_manager.h](#).

8.45.4.2 timestamp

```
uint32_t timestamp
```

Unix timestamp or system time.

Definition at line [4](#) of file [event_manager.h](#).

8.45.4.3 group

```
uint8_t group
```

Event group identifier.

Definition at line [6](#) of file [event_manager.h](#).

8.45.4.4 event

```
uint8_t event
```

Specific event identifier.

Definition at line [8](#) of file [event_manager.h](#).

8.46 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"
0010
0011 #define EVENT_BUFFER_SIZE 10
0012 #define EVENT_LOG_FILE "/event_log.csv"
0020
0021
0026 enum class EventGroup : uint8_t {
0028     SYSTEM    = 0x00,
0030     POWER     = 0x01,
0032     COMMS     = 0x02,
0034     GPS       = 0x03,
0036     CLOCK     = 0x04
0037 };
0038
0043 enum class SystemEvent : uint8_t {
0045     BOOT       = 0x01,
0047     SHUTDOWN   = 0x02,
0049     WATCHDOG_RESET = 0x03,
0051     CORE1_START = 0x04,
0053     CORE1_STOP  = 0x05
0054 };
0055
0060 enum class PowerEvent : uint8_t {
0062     LOW_BATTERY      = 0x01,
0064     OVERCHARGE       = 0x02,
0066     POWER_FALLING   = 0x03,
0068     POWER_NORMAL     = 0x04,
0070     SOLAR_ACTIVE     = 0x05,
0072     SOLAR_INACTIVE   = 0x06,
0074     USB_CONNECTED    = 0x07,
0076     USB_DISCONNECTED = 0x08
0077 };
0078
0083 enum class CommsEvent : uint8_t {
0085     RADIO_INIT      = 0x01,
0087     RADIO_ERROR     = 0x02,
0089     MSG_RECEIVED    = 0x03,
0091     MSG_SENT        = 0x04,
0093     UART_ERROR      = 0x06
0094 };
0095
0100 enum class GPSEvent : uint8_t {
0102     LOCK           = 0x01,
0104     LOST            = 0x02,
0106     ERROR           = 0x03,
0108     POWER_ON         = 0x04,
0110     POWER_OFF        = 0x05,
0112     DATA_READY       = 0x06,
0114     PASS_THROUGH_START = 0x07,
0116     PASS_THROUGH_END = 0x08
0117 };
0118
0119
0124 enum class ClockEvent : uint8_t {
0126     CHANGED        = 0x01,
0128     GPS_SYNC        = 0x02
0129 };
0130
0131
0136 class EventLog {
0137     public:
0139         uint16_t id;
0141         uint32_t timestamp;
0143         uint8_t group;
0145         uint8_t event;
0146
0151         std::string to_string() const {
0152             char buffer[256];
0153             snprintf(buffer, sizeof(buffer),
0154                     "EventLog: id=%u, timestamp=%lu, group=%u, event=%u",
0155                     id, timestamp, group, event);
0156             return std::string(buffer);
0157         }
0158     } __attribute__((packed));

```

```

00159
00160
00165 class EventManager {
00166     public:
00171         EventManager()
00172             : eventCount(0)
00173             , writeIndex(0)
00174             , nextEventId(0)
00175             , needsPersistence(false)
00176     {
00177         mutex_init(&eventMutex);
00178     }
00179
00183     virtual ~EventManager() = default;
00184
00189     virtual void init() {
00190         load_from_storage();
00191     }
00192
00198     void log_event(uint8_t group, uint8_t event);
00199
00205     const EventLog& get_event(size_t index) const;
00206
00211     size_t get_event_count() const { return eventCount; }
00212
00217     virtual bool save_to_storage() = 0;
00218
00223     virtual bool load_from_storage() = 0;
00224
00225     protected:
00227         EventLog events[EVENT_BUFFER_SIZE];
00229         size_t eventCount;
00231         size_t writeIndex;
00233         mutex_t eventMutex;
00235         volatile uint16_t nextEventId;
00237         bool needsPersistence;
00238     };
00239
00240
00245 class EventManagerImpl : public EventManager {
00246     public:
00251         EventManagerImpl() {
00252             init(); // Safe to call virtual functions here
00253         }
00254
00260         bool save_to_storage() override {
00261             if(!sd_card_mounted) {
00262                 bool status = fs_init();
00263                 if(!status) {
00264                     return false;
00265                 }
00266             }
00267             FILE *file = fopen(EVENT_LOG_FILE, "a");
00268             if (file) {
00269                 for (size_t i = 0; i < eventCount; i++) {
00270                     fprintf(file, "%u;%lu;%u;%u\n",
00271                         events[i].id,
00272                         events[i].timestamp,
00273                         events[i].group,
00274                         events[i].event
00275                     );
00276                 }
00277                 fclose(file);
00278                 needsPersistence = false;
00279                 uart_print("Events saved to storage", VerboseLevel::INFO);
00280                 return true;
00281             }
00282             return false;
00283         }
00284
00290         bool load_from_storage() override {
00291             // TODO: Implement based on chosen storage (SD/EEPROM)
00292             return false;
00293         }
00294     };
00295
00296
00300 extern EventManagerImpl eventManager;
00301
00306 class EventEmitter {
00307     public:
00314     template<typename T>
00315     static void emit(EventGroup group, T event) {
00316         eventManager.log_event(
00317             static_cast<uint8_t>(group),
00318             static_cast<uint8_t>(event)
00319         );

```

```

00320      }
00321  };
00322
00323
00328 void check_power_events(PowerManager& pm);
00329
00330
00331 #endif // End of EventManagerGroup

```

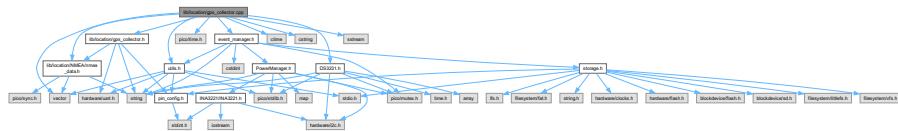
8.47 lib/location/gps_collector.cpp File Reference

```

#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 <sstream>

```

Include dependency graph for gps_collector.cpp:



Macros

- #define MAX_RAW_DATA_LENGTH 1024

Functions

- std::vector< std::string > splitString (const std::string &str, char delimiter)
- void collect_gps_data ()

Variables

- NMEAData nmea_data

8.47.1 Macro Definition Documentation

8.47.1.1 MAX_RAW_DATA_LENGTH

```
#define MAX_RAW_DATA_LENGTH 1024
```

Definition at line 13 of file [gps_collector.cpp](#).

8.47.2 Function Documentation

8.47.2.1 splitString()

```
std::vector< std::string > splitString (
    const std::string & str,
    char delimiter)
```

Definition at line 17 of file [gps_collector.cpp](#).

Here is the caller graph for this function:

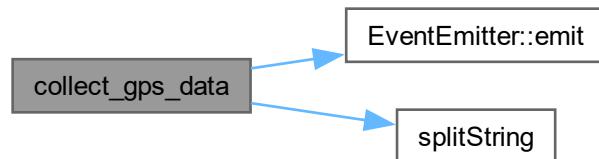


8.47.2.2 collect_gps_data()

```
void collect_gps_data ()
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.47.3 Variable Documentation

8.47.3.1 nmea_data

```
NMEAData nmea_data [extern]
```

Definition at line 3 of file NMEA_data.cpp.

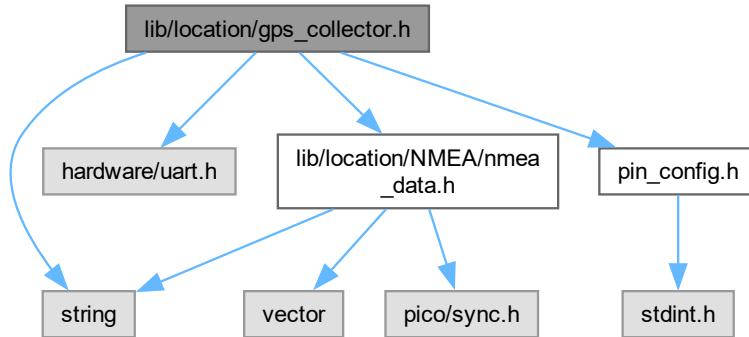
8.48 gps_collector.cpp

[Go to the documentation of this file.](#)

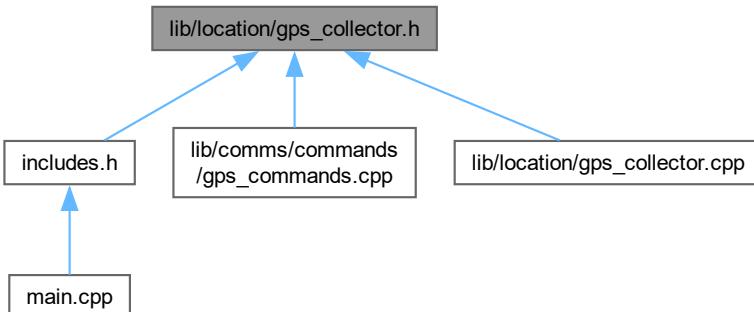
```
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/gps_collector.cpp
00002 #include "lib/location/gps_collector.h"
00003 #include "utils.h"
00004 #include "pico/time.h"
00005 #include "lib/location/NMEA/nmea_data.h"
00006 #include "event_manager.h"
00007 #include <vector>
00008 #include <ctime>
00009 #include <cstring>
00010 #include "DS3231.h"
00011 #include <sstream>
00012
00013 #define MAX_RAW_DATA_LENGTH 1024
00014
00015 extern NMEAData nmea_data;
00016
00017 std::vector<std::string> splitString(const std::string& str, char delimiter) {
00018     std::vector<std::string> tokens;
00019     std::stringstream ss(str);
00020     std::string token;
00021     while (std::getline(ss, token, delimiter)) {
00022         tokens.push_back(token);
00023     }
00024     return tokens;
00025 }
00026
00027 void collect_gps_data() {
00028     static char raw_data_buffer[MAX_RAW_DATA_LENGTH];
00029     static int raw_data_index = 0;
00030
00031     while (uart_is_readable(GPS_UART_PORT)) {
00032         char c = uart_getc(GPS_UART_PORT);
00033
00034         if (c == '\r' || c == '\n') {
00035             // End of message
00036             if (raw_data_index > 0) {
00037                 raw_data_buffer[raw_data_index] = '\0';
00038                 std::string message(raw_data_buffer);
00039                 raw_data_index = 0;
00040
00041                 // Split the message into tokens
00042                 std::vector<std::string> tokens = splitString(message, ',');
00043
00044                 // Update the global vectors based on the sentence type
00045                 if (message.find("$GPRMC") == 0) {
00046                     nmea_data.update_rmc_tokens(tokens);
00047                     EventEmitter::emit(EventGroup::GPS, GPSEvent::DATA_READY);
00048                 } else if (message.find("$GPGGA") == 0) {
00049                     nmea_data.update_gga_tokens(tokens);
00050                     EventEmitter::emit(EventGroup::GPS, GPSEvent::DATA_READY);
00051                 }
00052             }
00053         } else {
00054             // Append to buffer
00055             if (raw_data_index < MAX_RAW_DATA_LENGTH - 1) {
00056                 raw_data_buffer[raw_data_index++] = c;
00057             } else {
00058                 raw_data_index = 0;
00059             }
00060         }
00061     }
00062 }
```

8.49 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\(\)](#)

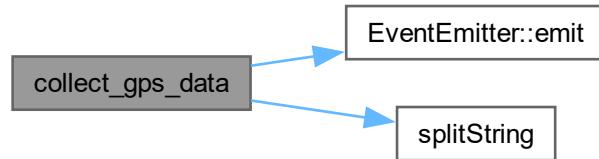
8.49.1 Function Documentation

8.49.1.1 [collect_gps_data\(\)](#)

```
void collect_gps_data ()
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



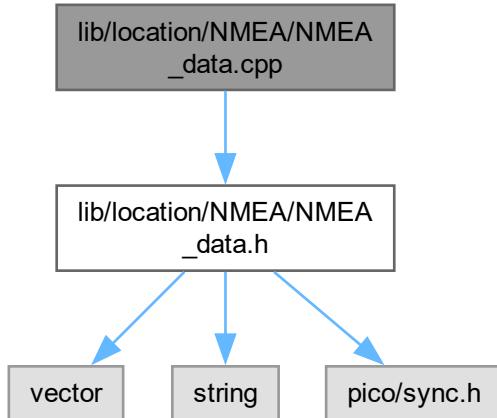
8.50 gps_collector.h

[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
```

8.51 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](#)

8.51.1 Variable Documentation

8.51.1.1 [nmea_data](#)

[NMEAData nmea_data](#)

Definition at line 3 of file [NMEA_data.cpp](#).

8.52 NMEA_data.cpp

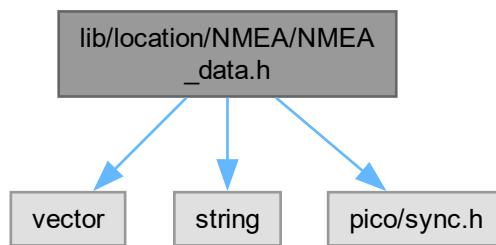
[Go to the documentation of this file.](#)

```
00001 #include "lib/location/NMEA/NMEA_data.h"
00002
00003 NMEAData nmea_data; // Define the global instance
00004
00005 NMEAData::NMEAData() {
00006     mutex_init(&rmc_mutex_);
00007     mutex_init(&gga_mutex_);
00008 }
00009
00010 void NMEAData::update_rmc_tokens(const std::vector<std::string>& tokens) {
00011     mutex_enter_blocking(&rmc_mutex_);
00012     rmc_tokens_ = tokens;
00013     mutex_exit(&rmc_mutex_);
00014 }
```

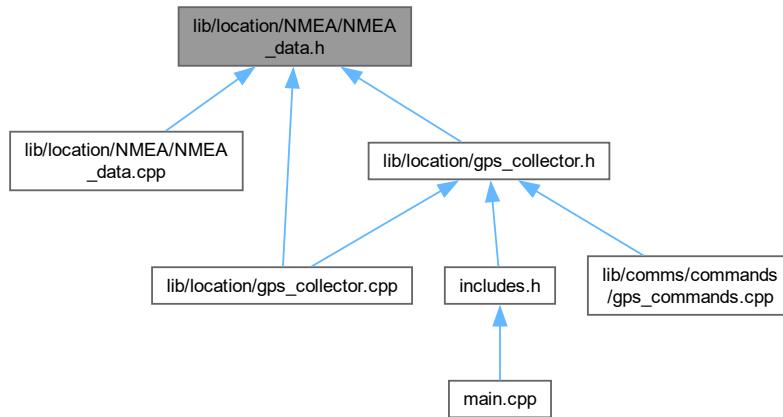
```
00015
00016 void NMEAData::update_gga_tokens(const std::vector<std::string>& tokens) {
00017     mutex_enter_blocking(&gga_mutex_);
00018     gga_tokens_ = tokens;
00019     mutex_exit(&gga_mutex_);
00020 }
00021
00022 std::vector<std::string> NMEAData::get_rmc_tokens() const {
00023     mutex_enter_blocking(const_cast<mutex_t*>(&rmc_mutex_));
00024     std::vector<std::string> copy = rmc_tokens_;
00025     mutex_exit(const_cast<mutex_t*>(&rmc_mutex_));
00026     return copy;
00027 }
00028
00029 std::vector<std::string> NMEAData::get_gga_tokens() const {
00030     mutex_enter_blocking(const_cast<mutex_t*>(&gga_mutex_));
00031     std::vector<std::string> copy = gga_tokens_;
00032     mutex_exit(const_cast<mutex_t*>(&gga_mutex_));
00033     return copy;
00034 }
```

8.53 lib/location/NMEA/NMEA_data.h File Reference

```
#include <vector>
#include <string>
#include "pico/sync.h"
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](#)

8.53.1 Variable Documentation

8.53.1.1 nmea_data

`NMEAData nmea_data [extern]`

Definition at line 3 of file [NMEA_data.cpp](#).

8.54 NMEA_data.h

[Go to the documentation of this file.](#)

```

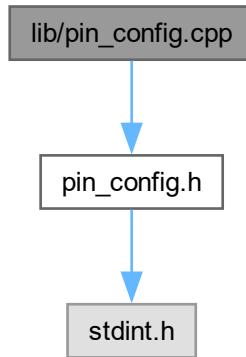
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/nmea_data.h
00002 #ifndef NMEA_DATA_H
00003 #define NMEA_DATA_H
00004
00005 #include <vector>
00006 #include <string>
00007 #include "pico/sync.h"
00008
00009 class NMEAData {
00010 public:
00011     NMEAData();
00012     void update_rmc_tokens(const std::vector<std::string>& tokens);
00013     void update_gga_tokens(const std::vector<std::string>& tokens);
00014     std::vector<std::string> get_rmc_tokens() const;
  
```

```
00016     std::vector<std::string> get_gga_tokens() const;
00017
00018 private:
00019     std::vector<std::string> rmc_tokens_;
00020     std::vector<std::string> gga_tokens_;
00021     mutex_t rmc_mutex_;
00022     mutex_t gga_mutex_;
00023 };
00024
00025 extern NMEAData nmea_data;
00026
00027 #endif
```

8.55 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

8.55.1 Variable Documentation

8.55.1.1 lora_cs_pin

```
const int lora_cs_pin = 17
```

Definition at line 4 of file [pin_config.cpp](#).

8.55.1.2 lora_reset_pin

```
const int lora_reset_pin = 22
```

Definition at line [5](#) of file [pin_config.cpp](#).

8.55.1.3 lora_irq_pin

```
const int lora_irq_pin = 28
```

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

8.55.1.4 lora_address_local

```
uint8_t lora_address_local = 37
```

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

8.55.1.5 lora_address_remote

```
uint8_t lora_address_remote = 21
```

Definition at line [9](#) of file [pin_config.cpp](#).

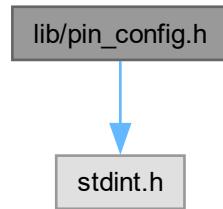
8.56 pin_config.cpp

[Go to the documentation of this file.](#)

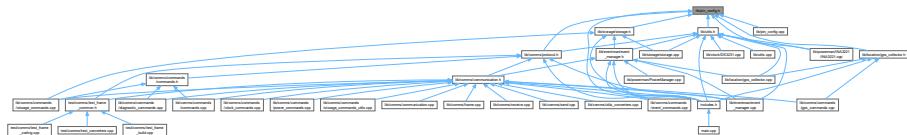
```
00001 #include "pin_config.h"
00002
00003 // LoRa constants
00004 const int lora_cs_pin = 17;           // LoRa radio chip select
00005 const int lora_reset_pin = 22;        // LoRa radio reset
00006 const int lora_irq_pin = 28;          // LoRa hardware interrupt pin
00007
00008 uint8_t lora_address_local = 37;      // address of this device
00009 uint8_t lora_address_remote = 21;
```

8.57 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

8.57.1 Macro Definition Documentation

8.57.1.1 DEBUG_UART_PORT

```
#define DEBUG_UART_PORT uart0
```

Definition at line 8 of file [pin_config.h](#).

8.57.1.2 DEBUG_UART_BAUD_RATE

```
#define DEBUG_UART_BAUD_RATE 115200
```

Definition at line 9 of file [pin_config.h](#).

8.57.1.3 DEBUG_UART_TX_PIN

```
#define DEBUG_UART_TX_PIN 0
```

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

8.57.1.4 DEBUG_UART_RX_PIN

```
#define DEBUG_UART_RX_PIN 1
```

Definition at line 12 of file [pin_config.h](#).

8.57.1.5 MAIN_I2C_PORT

```
#define MAIN_I2C_PORT i2c1
```

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

8.57.1.6 MAIN_I2C_SDA_PIN

```
#define MAIN_I2C_SDA_PIN 6
```

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

8.57.1.7 MAIN_I2C_SCL_PIN

```
#define MAIN_I2C_SCL_PIN 7
```

Definition at line 16 of file [pin_config.h](#).

8.57.1.8 GPS_UART_PORT

```
#define GPS_UART_PORT uart1
```

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

8.57.1.9 GPS_UART_BAUD_RATE

```
#define GPS_UART_BAUD_RATE 9600
```

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

8.57.1.10 GPS_UART_TX_PIN

```
#define GPS_UART_TX_PIN 8
```

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

8.57.1.11 GPS_UART_RX_PIN

```
#define GPS_UART_RX_PIN 9
```

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

8.57.1.12 GPS_POWER_ENABLE_PIN

```
#define GPS_POWER_ENABLE_PIN 14
```

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

8.57.1.13 BUFFER_SIZE

```
#define BUFFER_SIZE 85
```

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

8.57.1.14 SD_SPI_PORT

```
#define SD_SPI_PORT spil
```

Definition at line 28 of file [pin_config.h](#).

8.57.1.15 SD_MISO_PIN

```
#define SD_MISO_PIN 12
```

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

8.57.1.16 SD_MOSI_PIN

```
#define SD_MOSI_PIN 11
```

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

8.57.1.17 SD_SCK_PIN

```
#define SD_SCK_PIN 10
```

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

8.57.1.18 SD_CS_PIN

```
#define SD_CS_PIN 13
```

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

8.57.1.19 SD_CARD_DETECT_PIN

```
#define SD_CARD_DETECT_PIN 28
```

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

8.57.1.20 SX1278_MISO

```
#define SX1278_MISO 16
```

Definition at line 35 of file [pin_config.h](#).

8.57.1.21 SX1278_CS

```
#define SX1278_CS 17
```

Definition at line 36 of file [pin_config.h](#).

8.57.1.22 SX1278_SCK

```
#define SX1278_SCK 18
```

Definition at line [37](#) of file [pin_config.h](#).

8.57.1.23 SX1278_MOSI

```
#define SX1278_MOSI 19
```

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

8.57.1.24 SPI_PORT

```
#define SPI_PORT spio
```

Definition at line [40](#) of file [pin_config.h](#).

8.57.1.25 READ_BIT

```
#define READ_BIT 0x80
```

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

8.57.1.26 LORA_DEFAULT_SPI

```
#define LORA_DEFAULT_SPI spio
```

Definition at line [43](#) of file [pin_config.h](#).

8.57.1.27 LORA_DEFAULT_SPI_FREQUENCY

```
#define LORA_DEFAULT_SPI_FREQUENCY 8E6
```

Definition at line [44](#) of file [pin_config.h](#).

8.57.1.28 LORA_DEFAULT_SS_PIN

```
#define LORA_DEFAULT_SS_PIN 17
```

Definition at line [45](#) of file [pin_config.h](#).

8.57.1.29 LORA_DEFAULT_RESET_PIN

```
#define LORA_DEFAULT_RESET_PIN 22
```

Definition at line [46](#) of file [pin_config.h](#).

8.57.1.30 LORA_DEFAULT_DIO0_PIN

```
#define LORA_DEFAULT_DIO0_PIN 20
```

Definition at line 47 of file [pin_config.h](#).

8.57.1.31 PA_OUTPUT_RFO_PIN

```
#define PA_OUTPUT_RFO_PIN 11
```

Definition at line 49 of file [pin_config.h](#).

8.57.1.32 PA_OUTPUT_PA_BOOST_PIN

```
#define PA_OUTPUT_PA_BOOST_PIN 12
```

Definition at line 50 of file [pin_config.h](#).

8.57.2 Variable Documentation

8.57.2.1 lora_cs_pin

```
const int lora_cs_pin [extern]
```

Definition at line 4 of file [pin_config.cpp](#).

8.57.2.2 lora_reset_pin

```
const int lora_reset_pin [extern]
```

Definition at line 5 of file [pin_config.cpp](#).

8.57.2.3 lora_irq_pin

```
const int lora_irq_pin [extern]
```

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

8.57.2.4 lora_address_local

```
uint8_t lora_address_local [extern]
```

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

8.57.2.5 lora_address_remote

```
uint8_t lora_address_remote [extern]
```

Definition at line 9 of file [pin_config.cpp](#).

8.58 pin_config.h

[Go to the documentation of this file.](#)

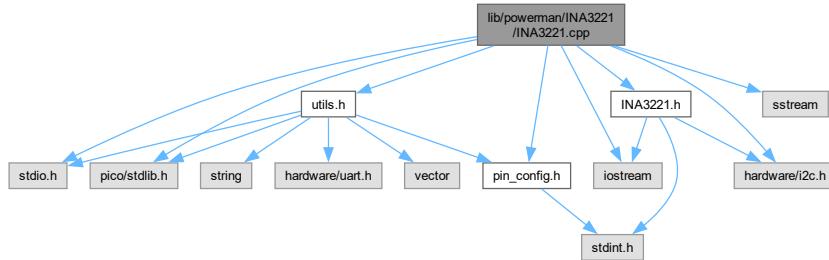
```
00001 // pin_config.h
00002 #include <stdint.h>
00003
00004 ifndef PIN_CONFIG_H
00005 define PIN_CONFIG_H
00006
00007 //DEBUG uart
00008 define DEBUG_UART_PORT uart0
00009 define DEBUG_UART_BAUD_RATE 115200
00010
00011 define DEBUG_UART_TX_PIN 0
00012 define DEBUG_UART_RX_PIN 1
00013
00014 define 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 spi1
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
```

8.59 lib/powerman/INA3221/INA3221.cpp File Reference

Implementation of the [INA3221](#) power monitor driver.

```
#include "INA3221.h"
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include <iostream>
#include "pin_config.h"
#include "utils.h"
#include <sstream>
```

Include dependency graph for INA3221.cpp:



8.59.1 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](#).

8.60 INA3221.cpp

[Go to the documentation of this file.](#)

```

00001 #include "INA3221.h"
00002 #include <stdio.h>
00003 #include "pico/stdlib.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(), VerbosityLevel::INFO);
00073
00074     if (manuf_id == 0x5449 && die_id == 0x3220) {
00075         uart_print("INA3221 found and initialized.", VerbosityLevel::DEBUG);
00076         return true;
00077     } else {
00078         uart_print("INA3221 initialization failed. Incorrect IDs.", VerbosityLevel::ERROR);
00079         return false;
00080     }
00081
00082 }
00083
00084
00085 00090 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
00095 00104 uint16_t INA3221::get_manufacturer_id() {
00096     uint16_t id = 0;
00097     _read(INA3221_REG_MANUF_ID, &id);
00098     return id;
00099 }
00100
00101
00102
00103 00116 uint16_t INA3221::get_die_id() {
00104     uint16_t id = 0;
00105     _read(INA3221_REG_DIE_ID, &id);
00106     return id;
00107 }
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136 //configure
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
```

```

00185 void INA3221::set_shunt_measurement_enable() {
00186     conf_reg_t conf_reg;
00187
00188     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00189     conf_reg.mode_shunt_en = 1;
00190     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00191 }
00192
00193
00198 void INA3221::set_shunt_measurement_disable() {
00199     conf_reg_t conf_reg;
00200
00201     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00202     conf_reg.mode_shunt_en = 0;
00203     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
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
00442 bool INA3221::get_warn_alert(ina3221_ch_t channel) {
00443     masken_reg_t masken;
00444     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00445
00446     switch(channel) {
00447         case INA3221_CH1: return masken.warn_alert_ch1;
00448         case INA3221_CH2: return masken.warn_alert_ch2;
00449         case INA3221_CH3: return masken.warn_alert_ch3;
00450         default: return false;
00451     }
00452 }
00453
00454
00455 bool INA3221::get_crit_alert(ina3221_ch_t channel) {
00456     masken_reg_t masken;
00457     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00458
00459     switch(channel) {
00460         case INA3221_CH1: return masken.crit_alert_ch1;
00461         case INA3221_CH2: return masken.crit_alert_ch2;
00462         case INA3221_CH3: return masken.crit_alert_ch3;
00463         default: return false;
00464     }
00465 }
00466
00467
00468 bool INA3221::get_power_valid_alert() {
00469     masken_reg_t masken;
00470     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00471     return masken.pwr_valid_alert;
00472 }
00473
00474
00475 void INA3221::set_alert_latch(bool enable) {
00476     masken_reg_t masken;
00477     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00478     masken.warn_alert_latch_en = enable;
00479     masken.crit_alert_latch_en = enable;
00480     _write(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00481 }
00482
00483
00484 // private
00485 void INA3221::_read(ina3221_reg_t reg, uint16_t *val) {
00486     uint8_t reg_buf = reg;
00487     uint8_t data[2];
00488
00489     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, &reg_buf, 1, true);
00490     if (ret != 1) {
00491         std::cerr << "Failed to write register address to I2C device." << std::endl;
00492         return;
00493     }
00494
00495     ret = i2c_read_blocking(MAIN_I2C_PORT, _i2c_addr, data, 2, false);
00496     if (ret != 2) {
00497         std::cerr << "Failed to read data from I2C device." << std::endl;
00498         return;
00499     }
00500
00501     *val = (data[0] << 8) | data[1];
00502 }
00503
00504
00505 void INA3221::_write(ina3221_reg_t reg, uint16_t *val) {
00506     uint8_t buf[3];
00507     buf[0] = reg;
00508     buf[1] = (*val >> 8) & 0xFF; // MSB
00509     buf[2] = (*val) & 0xFF; // LSB
00510
00511     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, buf, 3, false);
00512     if (ret != 3) {
00513         std::cerr << "Failed to write data to I2C device." << std::endl;
00514     }
00515 }
```

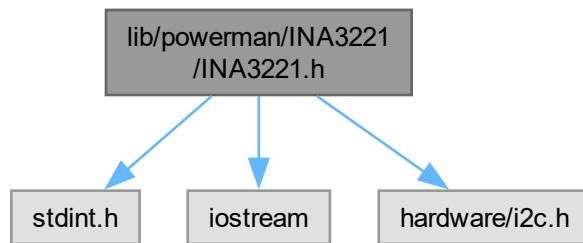
```
00548 }
00549 }
```

8.61 lib/powerman/INA3221/INA3221.h File Reference

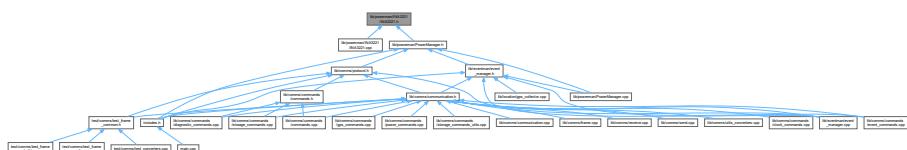
Header file for the [INA3221](#) triple-channel power monitor driver.

```
#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_ALEP` ,
`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.

8.61.1 Detailed Description

Header file for the `INA3221` triple-channel power monitor driver.

Definition in file `INA3221.h`.

8.61.2 Enumeration Type Documentation

8.61.2.1 `ina3221_addr_t`

```
enum ina3221_addr_t
```

Enumerator

INA3221_ADDR40_GND	
INA3221_ADDR41_VCC	
INA3221_ADDR42_SDA	
INA3221_ADDR43_SCL	

Definition at line 12 of file [INA3221.h](#).

8.61.2.2 ina3221_ch_t

```
enum ina3221_ch_t
```

Enumerator

INA3221_CH1	
INA3221_CH2	
INA3221_CH3	

Definition at line 23 of file [INA3221.h](#).

8.61.2.3 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](#).

8.61.2.4 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](#).

8.61.2.5 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](#).

8.61.3 Variable Documentation

8.61.3.1 INA3221_CH_NUM

const int INA3221_CH_NUM = 3

Number of channels in [INA3221](#).

Definition at line [30](#) of file [INA3221.h](#).

8.61.3.2 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](#).

8.62 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 continious
00171     void set_mode_continious();
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 mesurement
00180     void set_shunt_measurement_disable();
00181
00182     // Enables bus-voltage measurement
00183     void set_bus_measurement_enable();
00184
00185     // Disables bus-voltage measureement

```

```

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

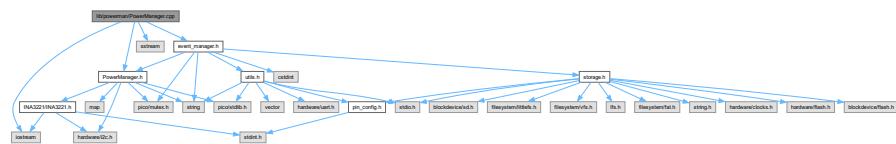
```

8.63 lib/powerman/PowerManager.cpp File Reference

```

#include "PowerManager.h"
#include <iostream>
#include <sstream>
#include "event_manager.h"
Include dependency graph for PowerManager.cpp:

```



8.64 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 };
0010
0011     bool PowerManager::initialize() {
0012         recursive_mutex_enter_blocking(&powerman_mutex_);
0013         initialized_ = ina3221_.begin();
0014
0015         if (initialized_) {
0016             // Set up alerts
0017             ina3221_.set_warn_alert_limit(INA3221_CH2, VOLTAGE_LOW_THRESHOLD);
0018             ina3221_.set_crit_alert_limit(INA3221_CH2, VOLTAGE_OVERCHARGE_THRESHOLD);
0019             ina3221_.set_power_valid_limit(VOLTAGE_OVERCHARGE_THRESHOLD, VOLTAGE_LOW_THRESHOLD);
0020             ina3221_.enable_alerts();
0021             ina3221_.set_alert_latch(true);
0022         }
0023
0024         recursive_mutex_exit(&powerman_mutex_);
0025         return initialized_;
0026     }
0027
0028     std::string PowerManager::read_device_ids() {
0029         if (!initialized_) return "noinit";
0030         recursive_mutex_enter_blocking(&powerman_mutex_);
0031         std::stringstream man_ss;
0032         man_ss << std::hex << ina3221_.get_manufacturer_id();
0033         std::string MAN = "MAN 0x" + man_ss.str();
0034
0035         std::stringstream die_ss;
0036         die_ss << std::hex << ina3221_.get_die_id();
0037         std::string DIE = "DIE 0x" + die_ss.str();
0038         recursive_mutex_exit(&powerman_mutex_);
0039         return MAN + " - " + DIE;
0040     }
0041
0042     float PowerManager::get_voltage_battery() {
0043         if (!initialized_) return 0.0f;
0044         recursive_mutex_enter_blocking(&powerman_mutex_);
0045         float voltage = ina3221_.get_voltage(INA3221_CH1);
0046         recursive_mutex_exit(&powerman_mutex_);
0047         return voltage;
0048     }
0049
0050     float PowerManager::get_voltage_5v() {
0051         if (!initialized_) return 0.0f;
0052         recursive_mutex_enter_blocking(&powerman_mutex_);
0053         float voltage = ina3221_.get_voltage(INA3221_CH2);
0054         recursive_mutex_exit(&powerman_mutex_);
0055         return voltage;
0056     }
0057
0058     float PowerManager::get_current_charge_usb() {
0059         if (!initialized_) return 0.0f;
0060         recursive_mutex_enter_blocking(&powerman_mutex_);
0061         float current = ina3221_.get_current_ma(INA3221_CH1);
0062         recursive_mutex_exit(&powerman_mutex_);
0063         return current;
0064     }
0065
0066     float PowerManager::get_current_draw() {
0067         if (!initialized_) return 0.0f;
0068         recursive_mutex_enter_blocking(&powerman_mutex_);
0069         float current = ina3221_.get_current_ma(INA3221_CH2);
0070         recursive_mutex_exit(&powerman_mutex_);
0071         return current;
0072     }
0073
0074     float PowerManager::get_current_charge_solar() {
0075         if (!initialized_) return 0.0f;
0076         recursive_mutex_enter_blocking(&powerman_mutex_);
0077         float current = ina3221_.get_current_ma(INA3221_CH3);
0078         recursive_mutex_exit(&powerman_mutex_);
0079         return current;
0080     }
0081
0082     float PowerManager::get_current_charge_total() {
0083         if (!initialized_) return 0.0f;
0084         recursive_mutex_enter_blocking(&powerman_mutex_);
0085         float current = ina3221_.get_current_ma(INA3221_CH1) + ina3221_.get_current_ma(INA3221_CH3);
0086         recursive_mutex_exit(&powerman_mutex_);
0087         return current;
0088     }
0089
0090     void PowerManager::configure(const std::map<std::string, std::string>& config) {
0091         if (!initialized_) return;
0092         recursive_mutex_enter_blocking(&powerman_mutex_);
0093
0094         if (config.find("operating_mode") != config.end()) {

```

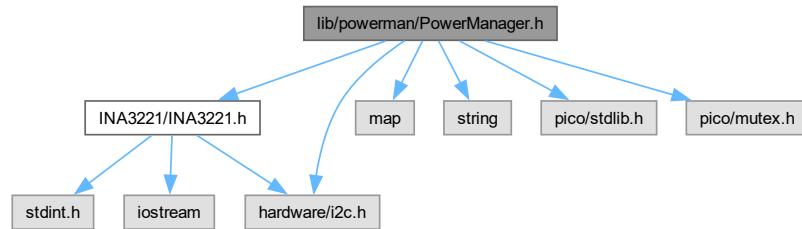
```

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             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 }
```

8.65 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](#)

8.66 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/stdcib.h"
00009 #include "pico/mutex.h"
00010
0011 class PowerManager {
0012 public:
0013     PowerManager(i2c_inst_t* i2c);
0014     bool initialize();
0015     std::string read_device_ids();
0016     float get_current_charge_solar();
0017     float get_current_charge_usb();
0018     float get_current_charge_total();
0019     float get_current_draw();
0020     float get_voltage_battery();
0021     float get_voltage_5v();
0022     void configure(const std::map<std::string, std::string>& config);
0023     bool is_charging_solar();
0024     bool is_charging_usb();
0025     bool check_power_alerts();
0026
0027
0028     static constexpr float SOLAR_CURRENT_THRESHOLD = 50.0f; // mA
0029     static constexpr float USB_CURRENT_THRESHOLD = 50.0f; // mA
0030     static constexpr float VOLTAGE_LOW_THRESHOLD = 4.6f; // V
0031     static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f; // V
0032     static constexpr float FALL_RATE_THRESHOLD = -0.02f; // V/sample

```

```

00033     static constexpr int FALLING_TREND_REQUIRED = 3;           // samples
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

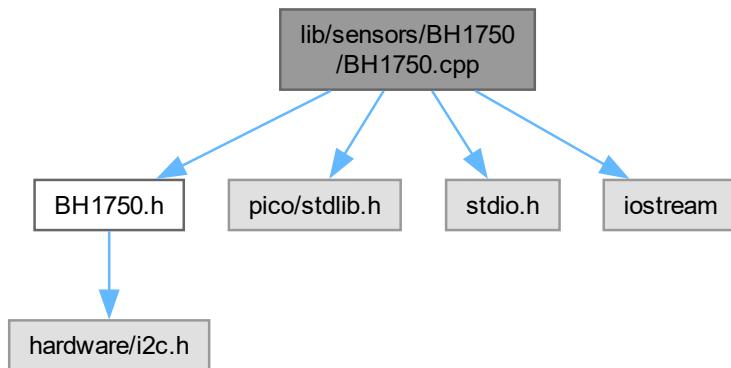
```

8.67 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:

```



8.68 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 }

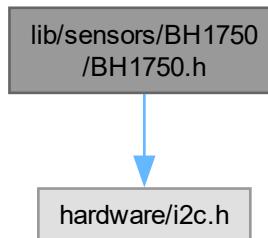
```

```
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 }
```

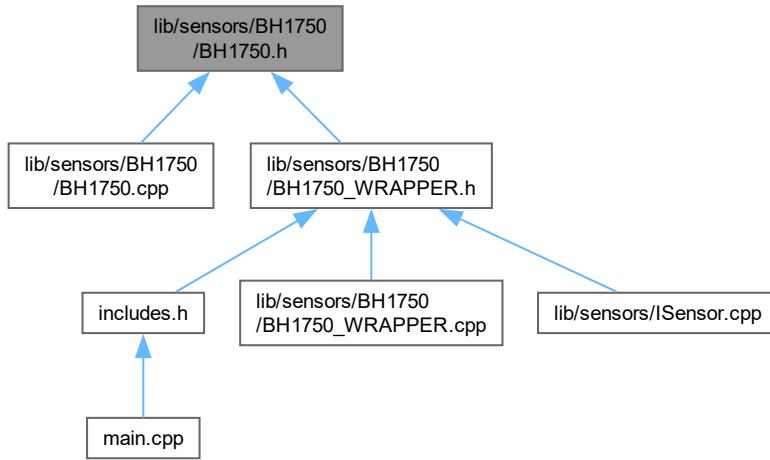
8.69 lib/sensors/BH1750/BH1750.h File Reference

#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`

8.69.1 Macro Definition Documentation

8.69.1.1 `_BH1750_DEVICE_ID`

```
#define _BH1750_DEVICE_ID 0xE1
```

Definition at line [7](#) of file [BH1750.h](#).

8.69.1.2 `_BH1750_MTREG_MIN`

```
#define _BH1750_MTREG_MIN 31
```

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

8.69.1.3 _BH1750_MTREG_MAX

```
#define _BH1750_MTREG_MAX 254
```

Definition at line 9 of file [BH1750.h](#).

8.69.1.4 _BH1750_DEFAULT_MTREG

```
#define _BH1750_DEFAULT_MTREG 69
```

Definition at line 10 of file [BH1750.h](#).

8.70 BH1750.h

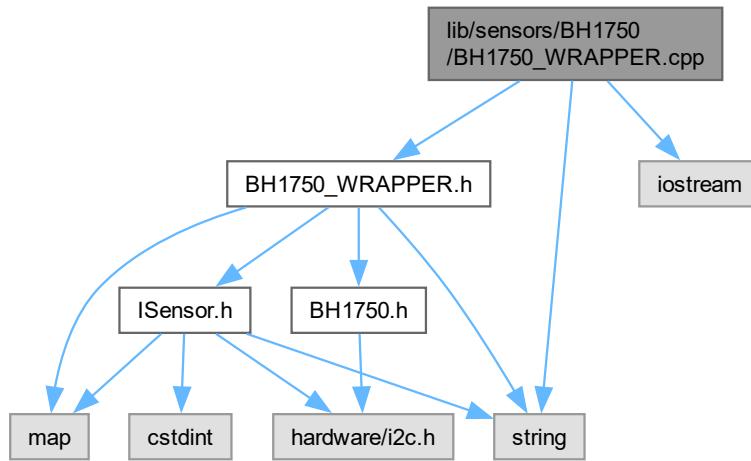
[Go to the documentation of this file.](#)

```
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__
```

8.71 lib/sensors/BH1750/BH1750_WRAPPER.cpp File Reference

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

```
#include <iostream>
Include dependency graph for BH1750_WRAPPER.cpp:
```



8.72 BH1750_WRAPPER.cpp

[Go to the documentation of this file.](#)

```

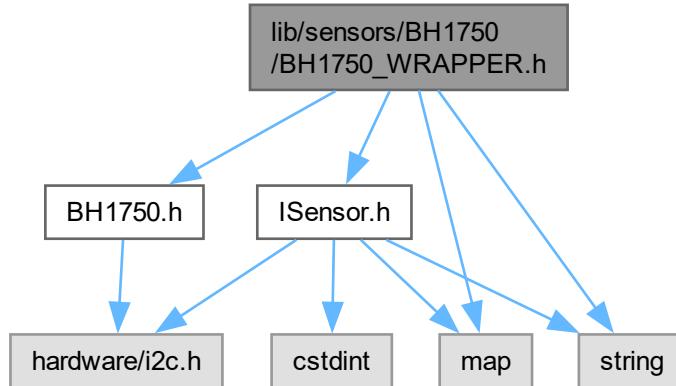
00001 // BH1750Wrapper.cpp
00002 #include "BH1750_WRAPPER.h"
00003 #include <string>
00004 #include <iostream>
00005
00006 BH1750Wrapper::BH1750Wrapper() {
00007     sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE);
00008 }
00009
00010 bool BH1750Wrapper::init() {
00011     initialized_ = sensor_.begin();
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         }
00043     }
00044 }
```

```

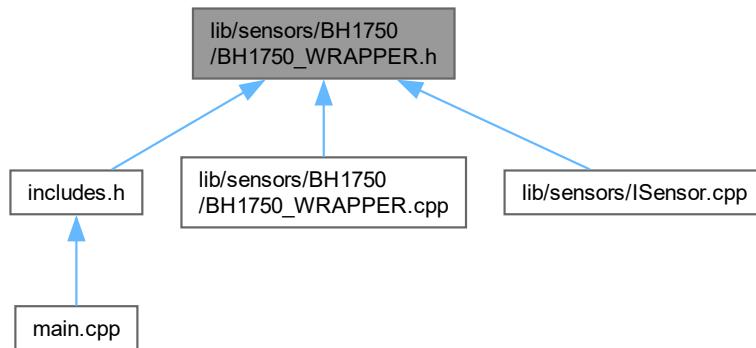
00042     else if (value == "one_time_high_resolution") {
00043         sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE);
00044     }
00045     else if (value == "one_time_high_resolution_2") {
00046         sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE_2);
00047     }
00048     else 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 }
00056 // Handle additional configuration keys here
00057 else {
00058     std::cerr << "[BH1750Wrapper] Unknown configuration key: " << key << std::endl;
00059     return false;
00060 }
00061 }
00062 return true;
00063 }
```

8.73 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](#)

8.74 BH1750_WRAPPER.h

[Go to the documentation of this file.](#)

```

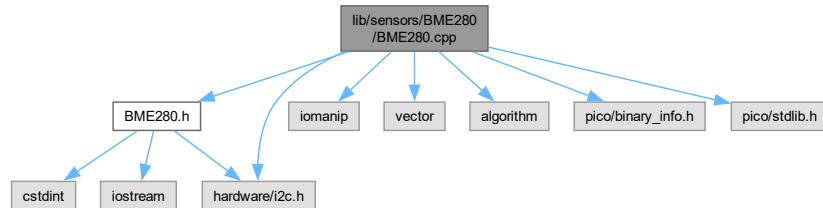
00001 #ifndef BH1750_WRAPPER_H
00002 #define BH1750_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "BH1750.h"
00006 #include <map>
00007 #include <string>
00008
00009 class BH1750Wrapper : public ISensor {
00010 private:
00011     BH1750 sensor_;
00012     bool initialized_ = false;
00013
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 };
00025
00026 #endif // BH1750_WRAPPER_H
  
```

8.75 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/stdlib.h"
Include dependency graph for BME280.cpp:
```



8.76 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/stdlib.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
  
```

```

00054     initialized_ = true;
00055     std::cout << "BME280 sensor initialized_ successfully.\n";
00056     return true;
00057 }
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 *
00140     (int32_t)calib_params.dig_h3) » 11) + 32768)) » 10) + 2097152) *

```

```

00136             (int32_t)calib_params.dig_h2 + 8192) » 14));
00137     v_xl_u32r = std::max(v_xl_u32r, (int32_t)0);
00138     v_xl_u32r = std::min(v_xl_u32r, (int32_t)419430400);
00139     float h = v_xl_u32r » 12;
00140     return h / 1024.0f;
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 xl6
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

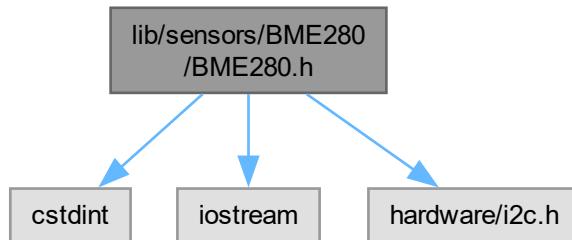
```

```

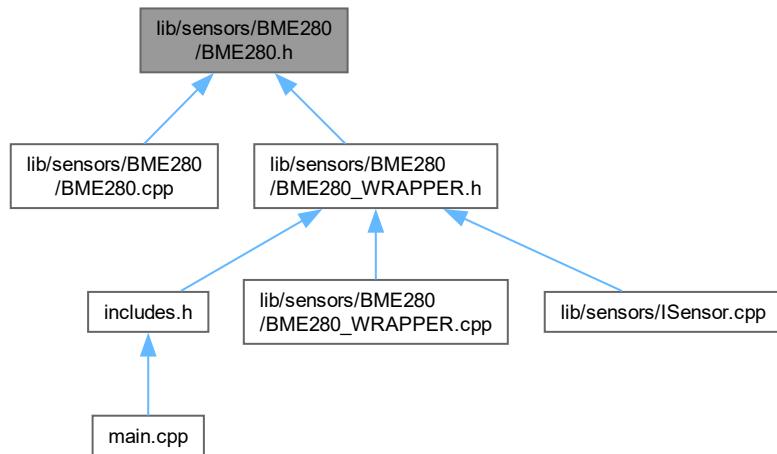
00223     buf[0] = REG_CTRL_MEAS;
00224     buf[1] = 0xB7; // Temp and pressure oversampling x16, normal mode
00225     ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00226     if (ret !=2) {
00227         std::cerr << "Failed to write CTRL_MEAS to BME280.\n";
00228         return false;
00229     }
00230
00231     return true;
00232 }
```

8.77 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**

8.78 BME280.h

[Go to the documentation of this file.](#)

```

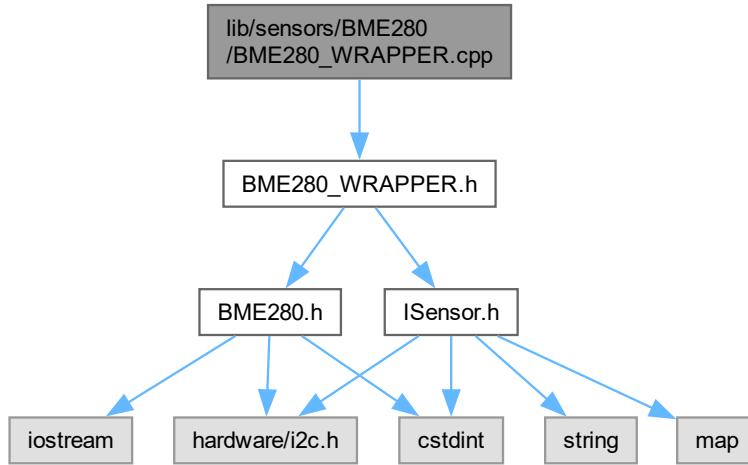
00001 // BME280.h
00002
00003 #ifndef BME280_H
00004 #define BME280_H
00005
00006 #include <cstdint>
00007 #include <iostream>
00008 #include "hardware/i2c.h"
00009
00010 // Calibration parameters structure
00011 struct BME280CalibParam {
00012     // Temperature parameters
00013     uint16_t dig_t1;
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
```

8.79 lib/sensors/BME280/BME280_WRAPPER.cpp File Reference

```
#include "BME280_WRAPPER.h"
Include dependency graph for BME280_WRAPPER.cpp:
```



8.80 BME280_WRAPPER.cpp

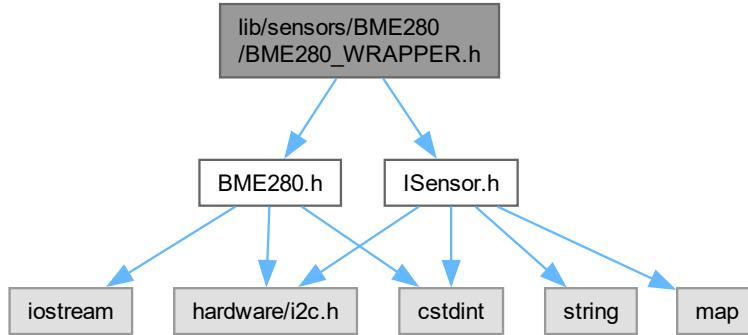
[Go to the documentation of this file.](#)

```

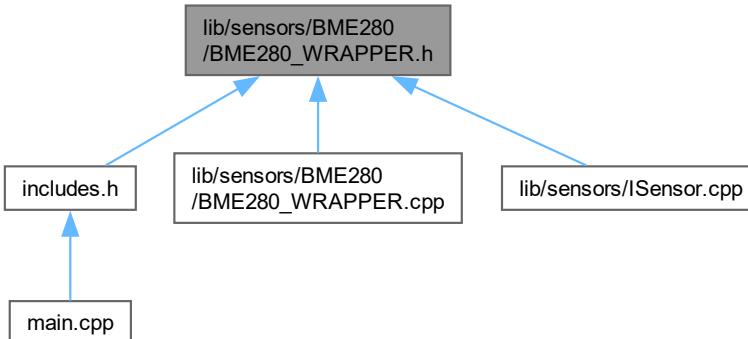
00001 #include "BME280_WRAPPER.h"
00002
00003 BME280Wrapper::BME280Wrapper(i2c_inst_t* i2c) : sensor_(i2c) {}
00004
00005 bool BME280Wrapper::init() {
00006     initialized_ = sensor_.init();
00007     return initialized_;
00008 }
00009
00010 float BME280Wrapper::read_data(SensorDataTypeIdentifier type) {
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 }
```

8.81 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](#)

8.82 BME280_WRAPPER.h

[Go to the documentation of this file.](#)

```
00001 // BME280_WRAPPER.h
00002 #ifndef BME280_WRAPPER_H
```

```

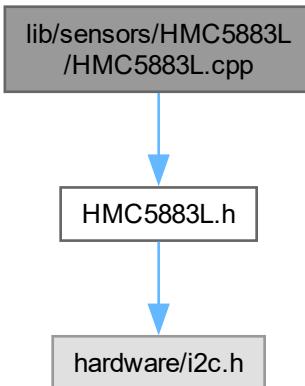
00003 #define BME280_WRAPPER_H
00004
00005 #include "ISensor.h"
00006 #include "BME280.h"
00007
00008 class BME280Wrapper : public ISensor {
00009 private:
0010     BME280 sensor_;
0011     bool initialized_ = false;
0012
0013 public:
0014     BME280Wrapper(i2c_inst_t* i2c);
0015
0016     bool init() 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 };
0023
0024 #endif // BME280_WRAPPER_H

```

8.83 lib/sensors/HMC5883L/HMC5883L.cpp File Reference

#include "HMC5883L.h"

Include dependency graph for HMC5883L.cpp:



8.84 HMC5883L.cpp

[Go to the documentation of this file.](#)

```

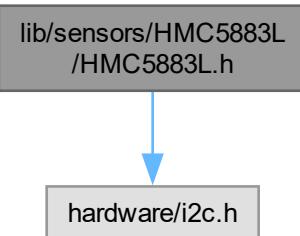
00001 #include "HMC5883L.h"
00002
00003 HMC5883L::HMC5883L(i2c_inst_t* i2c, uint8_t address) : i2c(i2c), address(address) {}
00004
00005 bool HMC5883L::init() {
00006     // Continuous measurement mode, 15Hz data output rate
00007     if (!write_register(0x00, 0x70)) return false;
00008     if (!write_register(0x01, 0x20)) return false;
00009     if (!write_register(0x02, 0x00)) return false;
00010     return true;
00011 }
00012

```

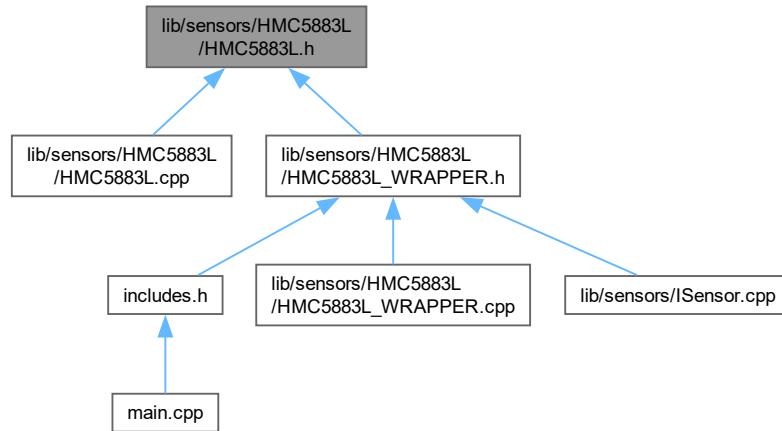
```
00013 bool HMC5883L::read(int16_t& x, int16_t& y, int16_t& z) {
00014     uint8_t buffer[6];
00015     if (!read_register(0x03, buffer, 6)) return false;
00016
00017     x = (buffer[0] << 8) | buffer[1];
00018     z = (buffer[2] << 8) | buffer[3];
00019     y = (buffer[4] << 8) | buffer[5];
00020
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 }
```

8.85 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](#)

8.86 HMC5883L.h

[Go to the documentation of this file.](#)

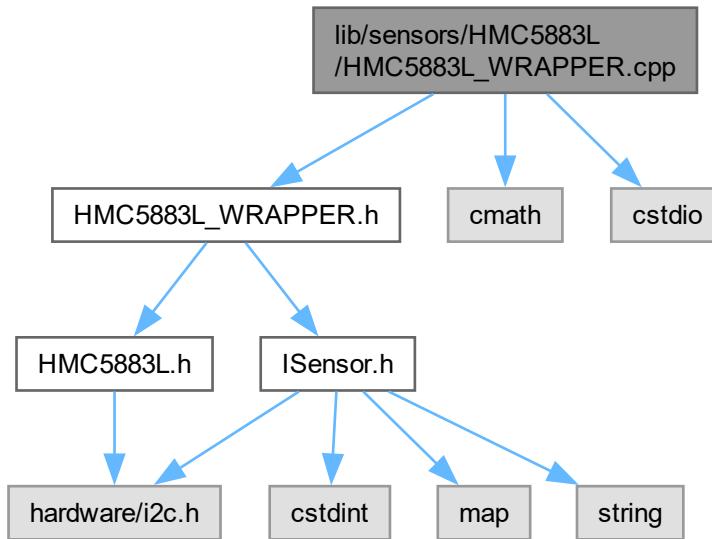
```

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
  
```

8.87 lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp File Reference

```
#include "HMC5883L_WRAPPER.h"
#include <cmath>
```

```
#include <cstdio>
Include dependency graph for HMC5883L_WRAPPER.cpp:
```



8.88 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_;
  
```

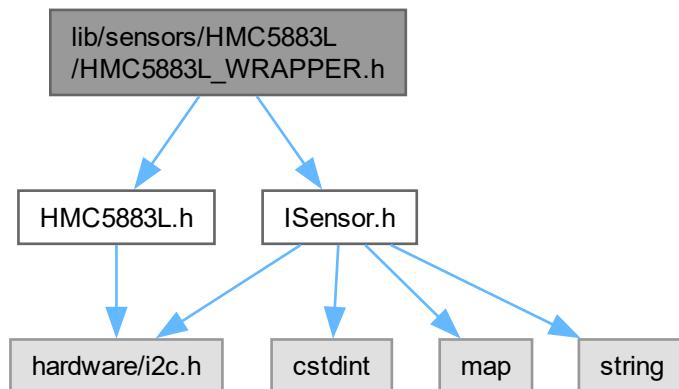
```

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 }

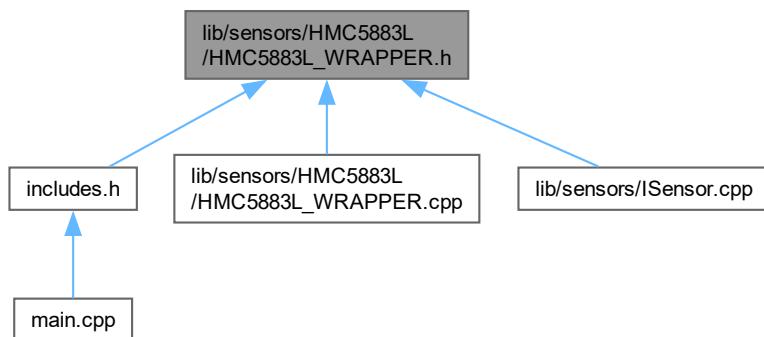
```

8.89 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](#)

8.90 HMC5883L_WRAPPER.h

[Go to the documentation of this file.](#)

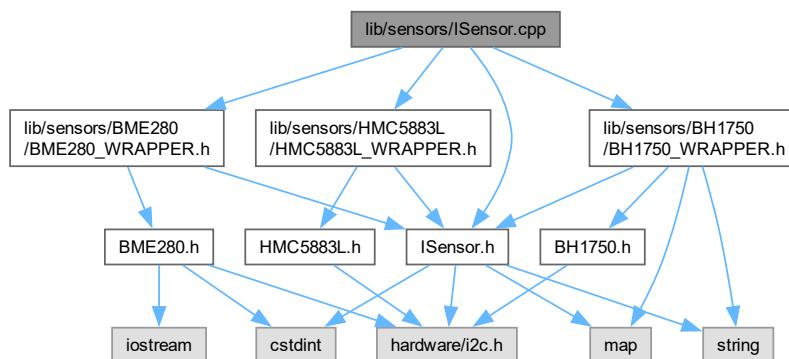
```
00001 #ifndef HMC5883L_WRAPPER_H
00002 #define HMC5883L_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "HMC5883L.h"
00006
00007 class HMC5883LWrapper : public ISensor {
00008 public:
00009     HMC5883LWrapper(i2c_inst_t* i2c);
00010     bool init() override;
00011     float read_data(SensorDataTypeIdentifier 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 private:
00017     HMC5883L sensor_;
00018     bool initialized_;
00019 };
00020
00021 #endif
```

8.91 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 dependency graph for ISensor.cpp:



8.91.1 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](#).

8.92 ISensor.cpp

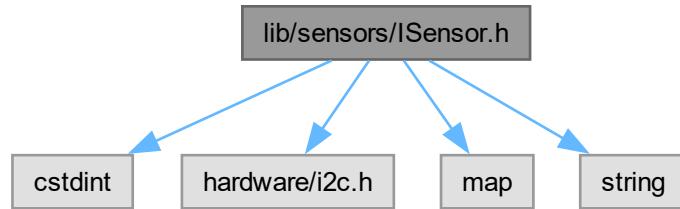
[Go to the documentation of this file.](#)

```
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
00013
00018
00023 SensorWrapper& SensorWrapper::get_instance() {
00024     static SensorWrapper instance;
00025     return instance;
00026 }
00027
00031 SensorWrapper::SensorWrapper() = default;
00032
00039 bool SensorWrapper::sensor_init(SensorType type, i2c_inst_t* i2c) {
00040     switch(type) {
00041         case SensorType::LIGHT:
00042             sensors[type] = new BH1750Wrapper();
00043             break;
00044         case SensorType::ENVIRONMENT:
00045             sensors[type] = new BME280Wrapper(i2c);
00046             break;
00047         case SensorType::IMU:
00048             //sensors[type] = new MPU6050Wrapper(i2c);
00049             break;
00050         case SensorType::MAGNETOMETER:
00051             sensors[type] = new HMC5883LWrapper(i2c);
00052             break;
00053     }
00054     return sensors[type]->init();
00055 }
00056
00063 bool SensorWrapper::sensor_configure(SensorType type, const std::map<std::string, std::string>& config) {
00064     auto it = sensors.find(type);
00065     if (it != sensors.end() && it->second->is_initialized()) {
00066         return it->second->configure(config);
00067     }
00068     std::cerr << "Sensor not initialized or not found: " << static_cast<int>(type) << std::endl;
00069     return false;
00070 }
00071
00078 float SensorWrapper::sensor_read_data(SensorType sensorType, SensorDataTypeIdentifier dataType) {
00079     auto it = sensors.find(sensorType);
00080     if (it != sensors.end() && it->second->is_initialized()) {
00081         return it->second->read_data(dataType);
00082     }
00083     return 0.0f;
00084 }
```

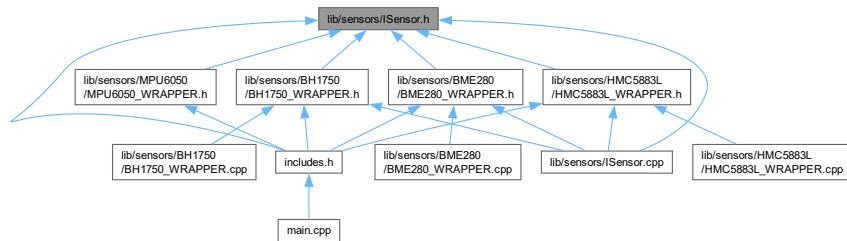
8.93 lib/sensors/ISensor.h File Reference

```
#include <cstdint>
#include "hardware/i2c.h"
#include <map>
```

```
#include <string>
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` }

8.93.1 Enumeration Type Documentation

8.93.1.1 SensorType

```
enum class SensorType [strong]
```

Enumerator

LIGHT	
ENVIRONMENT	
MAGNETOMETER	
IMU	

Definition at line 10 of file [ISensor.h](#).

8.93.1.2 SensorDataTypeIdentifier

```
enum class SensorDataTypeIdentifier [strong]
```

Enumerator

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	

Definition at line 17 of file [ISensor.h](#).

8.94 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
00010 enum class SensorType {
00011     LIGHT,           // BH1750
00012     ENVIRONMENT,    // BME280
00013     MAGNETOMETER,   // HMC5883L
00014     IMU,             // MPU6050
00015 };
00016
00017 enum class SensorDataTypeIdentifier {
00018     LIGHT_LEVEL,
00019     TEMPERATURE,
00020     PRESSURE,
00021     HUMIDITY,
00022     MAG_FIELD_X,
00023     MAG_FIELD_Y,
```

```
00024     MAG_FIELD_Z,  
00025     GYRO_X,  
00026     GYRO_Y,  
00027     GYRO_Z,  
00028     ACCEL_X,  
00029     ACCEL_Y,  
00030     ACCEL_Z  
00031 };  
00032  
00033 class ISensor {  
00034 public:  
00035     virtual ~ISensor() = default;  
00036     virtual bool init() = 0;  
00037     virtual float read_data(SensorDataTypeIdentifier type) = 0;  
00038     virtual bool is_initialized() const = 0;  
00039     virtual SensorType get_type() const = 0;  
00040     virtual bool configure(const std::map<std::string, std::string>& config) = 0;  
00041 };  
00042  
00043 class SensorWrapper {  
00044 public:  
00045     static SensorWrapper& get_instance();  
00046     bool sensor_init(SensorType type, i2c_inst_t* i2c = nullptr);  
00047     bool sensor_configure(SensorType type, const std::map<std::string, std::string>& config);  
00048     float sensor_read_data(SensorType sensorType, SensorDataTypeIdentifier dataType);  
00049  
00050 private:  
00051     std::map<SensorType, ISensor*> sensors;  
00052     SensorWrapper();  
00053 };  
00054  
00055 #endif // ISENSOR_H
```

8.95 lib/sensors/MPU6050/MPU6050.cpp File Reference

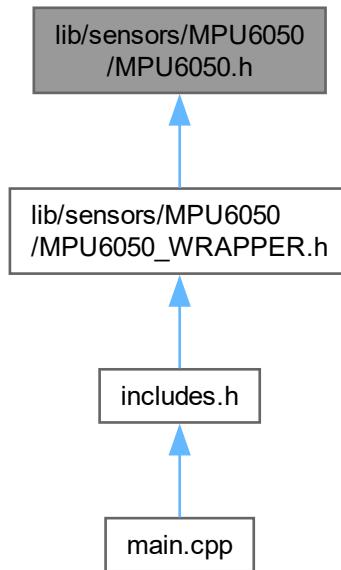
8.96 MPU6050.cpp

[Go to the documentation of this file.](#)

```
00001
```

8.97 lib/sensors/MPU6050/MPU6050.h File Reference

This graph shows which files directly or indirectly include this file:



8.98 MPU6050.h

[Go to the documentation of this file.](#)

00001

8.99 lib/sensors/MPU6050/MPU6050_WRAPPER.cpp File Reference

8.100 MPU6050_WRAPPER.cpp

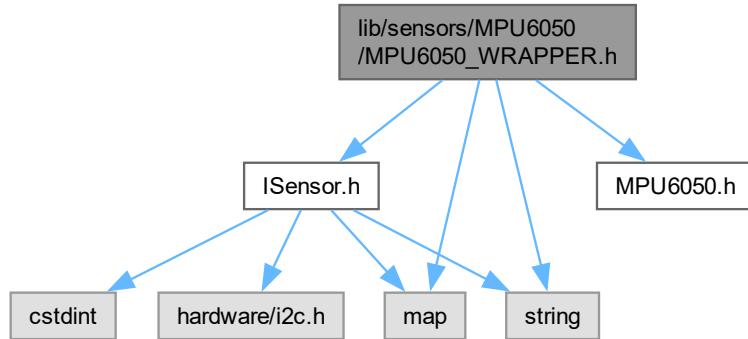
[Go to the documentation of this file.](#)

00001

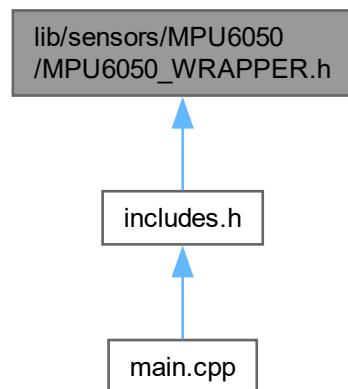
8.101 lib/sensors/MPU6050/MPU6050_WRAPPER.h File Reference

```
#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](#)

8.102 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

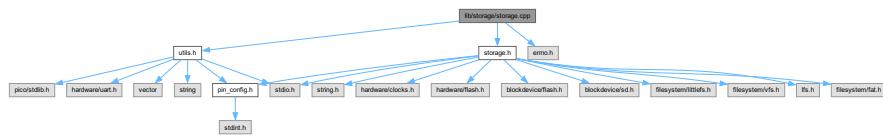
```

8.103 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`

8.103.1 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](#).

8.103.2 Function Documentation

8.103.2.1 fs_init()

```
bool fs_init (
    void )
```

Initializes the file system on the SD card.

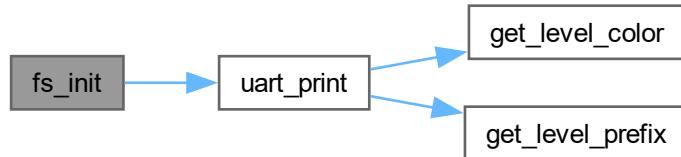
Returns

True if initialization was successful, false otherwise.

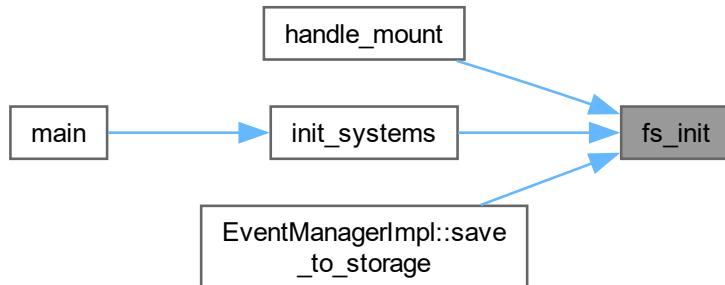
Mounts the littlefs file system on the SD card. If mounting fails, it formats the SD card with littlefs and then attempts to mount again.

Definition at line 25 of file [storage.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.103.3 Variable Documentation

8.103.3.1 sd_card_mounted

```
bool sd_card_mounted = false
```

Definition at line 17 of file [storage.cpp](#).

8.104 storage.cpp

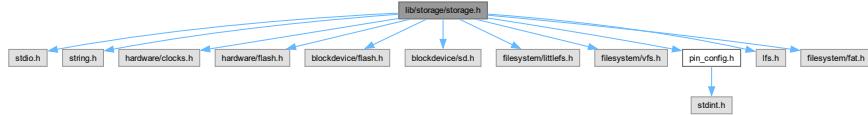
[Go to the documentation of this file.](#)

```
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
00016
00017 bool sd_card_mounted = false;
00018
00025 bool fs_init(void) {
00026     sd_card_mounted = false;
00027     uart_print("fs_init littlefs on SD card", VerbosityLevel::DEBUG);
00028     blockdevice_t *sd = blockdevice_sd_create(SD_SPI_PORT,
00029                                              SD_MOSI_PIN,
00030                                              SD_MISO_PIN,
00031                                              SD_SCK_PIN,
00032                                              SD_CS_PIN,
00033                                              24 * MHZ,
00034                                              false);
00035     filesystem_t *fat = filesystem_fat_create();
00036
00037     std::string statusString;
00038     int err = fs_mount("/", fat, sd);
00039     if (err == -1) {
00040         statusString = "Formatting / with FAT";
00041         uart_print(statusString, VerbosityLevel::WARNING);
00042         err = fs_format(fat, sd);
00043         if (err == -1) {
00044             statusString = "fs_format error: " + std::string(strerror(errno));
00045             uart_print(statusString, VerbosityLevel::ERROR);
00046             return false;
00047         }
00048         err = fs_mount("/", fat, sd);
00049         if (err == -1) {
00050             statusString = "fs_mount error: " + std::string(strerror(errno));
00051             uart_print(statusString, VerbosityLevel::ERROR);
00052             return false;
00053         }
00054     }
00055
00056     sd_card_mounted = true;
00057     return true;
00058 }
```

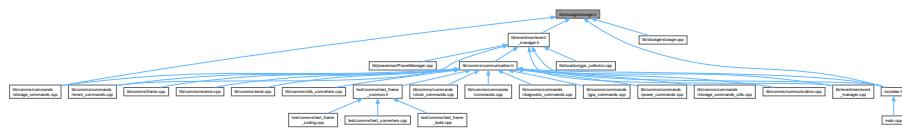
8.105 lib/storage/storage.h File Reference

```
#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](#)

8.105.1 Function Documentation

8.105.1.1 [fs_init\(\)](#)

```
bool fs_init (
    void )
```

Initializes the file system on the SD card.

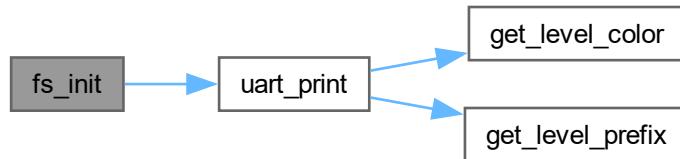
Returns

True if initialization was successful, false otherwise.

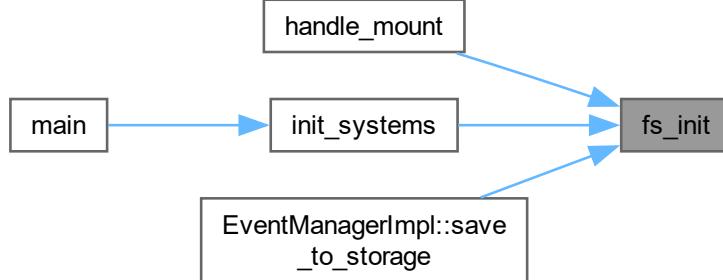
Mounts the littlefs file system on the SD card. If mounting fails, it formats the SD card with littlefs and then attempts to mount again.

Definition at line 25 of file [storage.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.105.1.2 `fs_open_file()`

```

FileHandle fs_open_file (
    const char * filename,
    const char * mode)
  
```

8.105.1.3 `fs_write_file()`

```

ssize_t fs_write_file (
    FileHandle & handle,
    const void * buffer,
    size_t size)
  
```

8.105.1.4 fs_read_file()

```
ssize_t fs_read_file (
    FileHandle & handle,
    void * buffer,
    size_t size)
```

8.105.1.5 fs_close_file()

```
bool fs_close_file (
    FileHandle & handle)
```

8.105.1.6 fs_file_exists()

```
bool fs_file_exists (
    const char * filename)
```

8.105.2 Variable Documentation**8.105.2.1 sd_card_mounted**

```
bool sd_card_mounted [extern]
```

Definition at line 17 of file [storage.cpp](#).

8.106 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
```

```

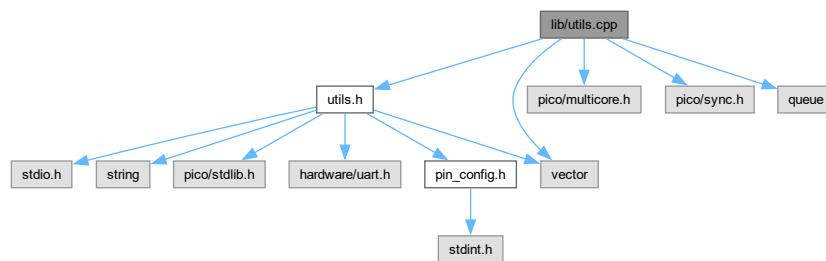
00032
00033 // void example_file_operations() {
00034 //     // Open a file for writing
00035 //     FileHandle log_file = fs_open_file("/log.txt", "w");
00036 //     if (!log_file.is_open) {
00037 //         uartPrint("Failed to open log file");
00038 //         return;
00039 //     }
00040
00041 //     // 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 // }

```

8.107 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 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.
- uint16_t `crc16` (const uint8_t *data, size_t length)
Calculates the CRC16 checksum of the given data.

Variables

- static mutex_t `uart_mutex`
Mutex for UART access protection.
- `VerbosityLevel g_uart_verbosity = VerbosityLevel::DEBUG`
Global verbosity level setting.

8.107.1 Detailed Description

Implementation of utility functions for the Kubisat firmware.

Definition in file [utils.cpp](#).

8.107.2 Function Documentation

8.107.2.1 `get_level_color()`

```
std::string get_level_color (
    VerbosityLevel level)
```

Gets ANSI color code for verbosity level.

Parameters

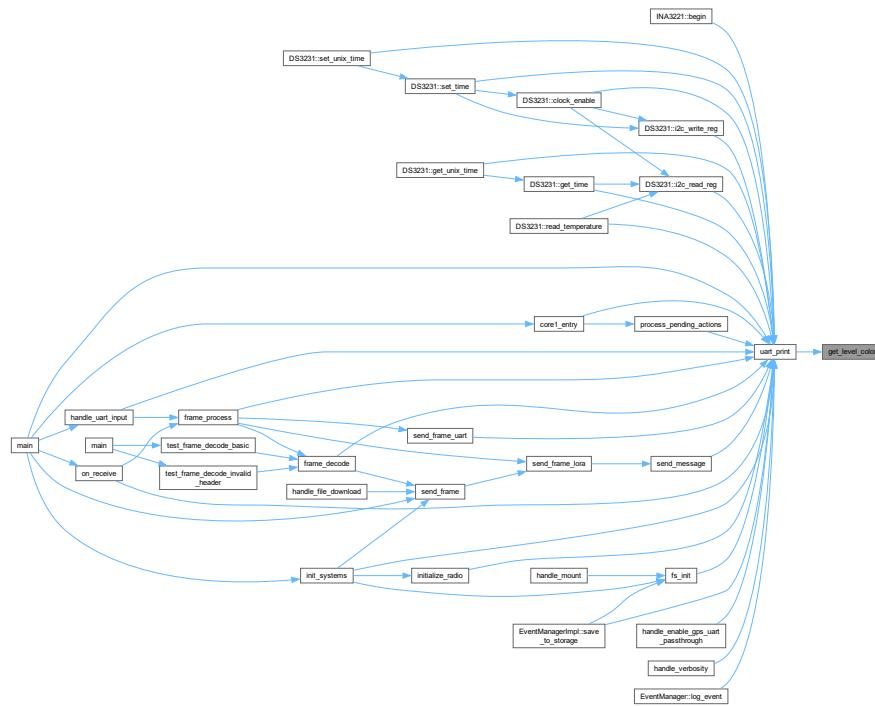
<code>level</code>	The verbosity level
--------------------	---------------------

Returns

ANSI color escape sequence

Definition at line 26 of file [utils.cpp](#).

Here is the caller graph for this function:



8.107.2.2 get_level_prefix()

```
std::string get_level_prefix (
    VerbosityLevel level)
```

Gets text prefix for verbosity level.

Parameters

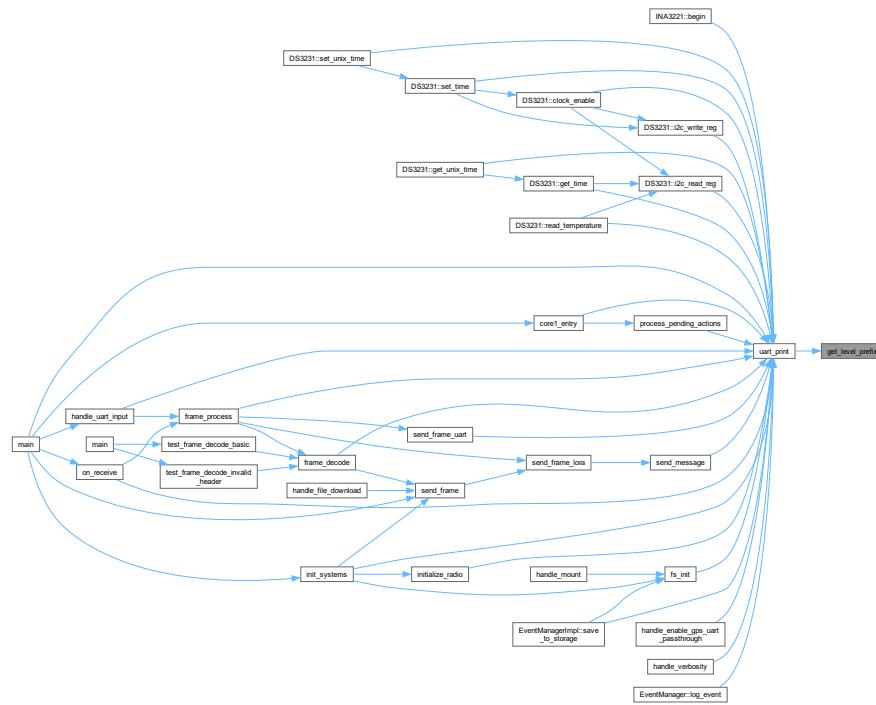
<i>level</i>	The verbosity level
--------------	---------------------

Returns

Text prefix for the level

Definition at line 43 of file [utils.cpp](#).

Here is the caller graph for this function:



8.107.2.3 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.

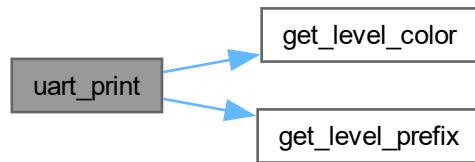
Parameters

<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.

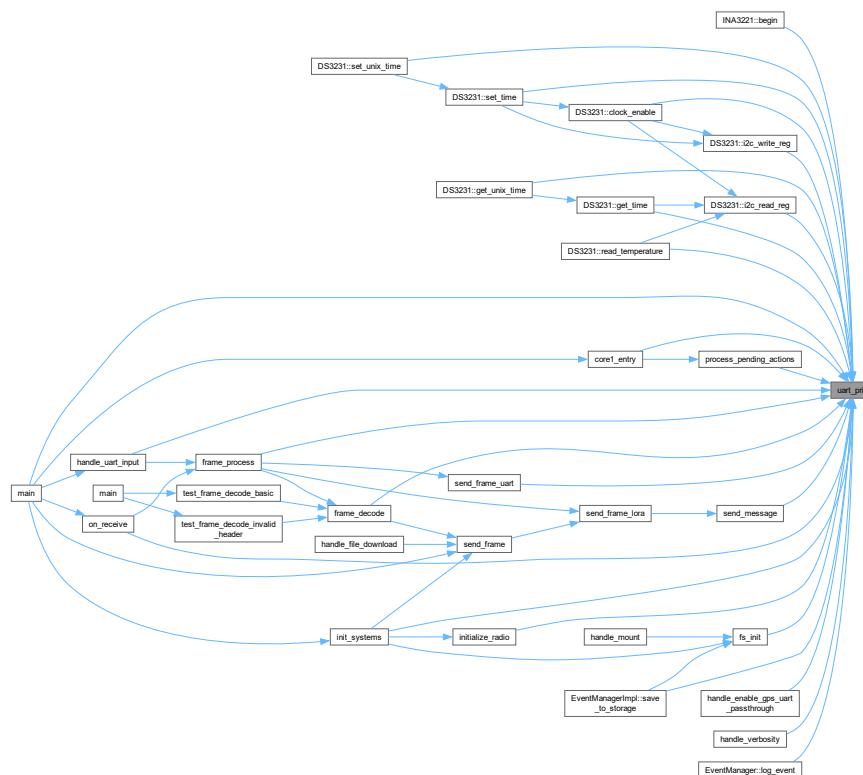
Prints the given message to the specified UART, prepending it with a timestamp and the core number. Uses a mutex to ensure thread-safe access to the UART.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.107.2.4 crc16()

```

uint16_t crc16 (
    const uint8_t * data,
    size_t length)
  
```

Calculates the CRC16 checksum of the given data.

Calculates CRC16 checksum.

Parameters

<i>data</i>	A pointer to the data buffer.
<i>length</i>	The length of the data in bytes.

Returns

The CRC16 checksum.

Calculates the CRC16 checksum using the standard algorithm.

Definition at line 97 of file [utils.cpp](#).

8.107.3 Variable Documentation

8.107.3.1 uart_mutex

```
mutex_t uart_mutex [static]
```

Mutex for UART access protection.

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

8.107.3.2 g_uart_verbosity

```
VerbosityLevel g_uart_verbosity = VerbosityLevel::DEBUG
```

Global verbosity level setting.

Global verbosity level setting for UART output.

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

8.108 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
00011
00012
00014 static mutex_t uart_mutex;
00015
00016
00018 VerbosityLevel g_uart_verbosity = VerbosityLevel::DEBUG;
00019
00020
00026 std::string get_level_color(VerbosityLevel level) {
00027     switch (level) {
00028         case VerbosityLevel::ERROR:    return ANSI_RED;
00029         case VerbosityLevel::WARNING: return ANSI_YELLOW;
00030         case VerbosityLevel::INFO:     return ANSI_GREEN;
00031         case VerbosityLevel::DEBUG:    return ANSI_BLUE;
```

```

00032     case VerboseLevel::EVENT:      return ANSI_CYAN;
00033     default:                      return "";
00034 }
00035 }
00036
00037
00043 std::string get_level_prefix(VerboseLevel level) {
00044     switch (level) {
00045         case VerboseLevel::ERROR:   return "ERROR: ";
00046         case VerboseLevel::WARNING: return "WARNING: ";
00047         case VerboseLevel::INFO:    return "INFO: ";
00048         case VerboseLevel::DEBUG:   return "DEBUG: ";
00049         case VerboseLevel::EVENT:   return "EVENT: ";
00050         default:                  return "";
00051     }
00052 }
00053
00062 void uart_print(const std::string& msg, VerboseLevel level, bool logToFile, uart_inst_t* uart) {
00063     if (static_cast<int>(level) > static_cast<int>(g_uart_verbosity)) {
00064         return;
00065     }
00066
00067     static bool mutex_initiated = false;
00068     if (!mutex_initiated) {
00069         mutex_init(&uart_mutex);
00070         mutex_initiated = true;
00071     }
00072
00073     uint32_t timestamp = to_ms_since_boot(get_absolute_time());
00074     uint core_num = get_core_num();
00075
00076     // Create formatted message with color
00077     std::string color = get_level_color(level);
00078     std::string prefix = get_level_prefix(level);
00079     std::string msgToSend = "[" + std::to_string(timestamp) + "ms] - Core " +
00080                           std::to_string(core_num) + ":" +
00081                           color + prefix + ANSI_RESET + msg + "\r\n";
00082
00083     // Print to UART
00084     mutex_enter_blocking(&uart_mutex);
00085     uart_puts(uart, msgToSend.c_str());
00086     mutex_exit(&uart_mutex);
00087 }
00088
00089
00097 uint16_t crc16(const uint8_t *data, size_t length) {
00098     uint16_t crc = 0xFFFF;
00099     for (size_t i = 0; i < length; i++) {
00100         crc ^= data[i];
00101         for (int j = 0; j < 8; j++) {
00102             if (crc & 0x0001) {
00103                 crc = (crc >> 1) ^ 0xA001;
00104             } else {
00105                 crc >>= 1;
00106             }
00107         }
00108     }
00109     return crc;
00110 }

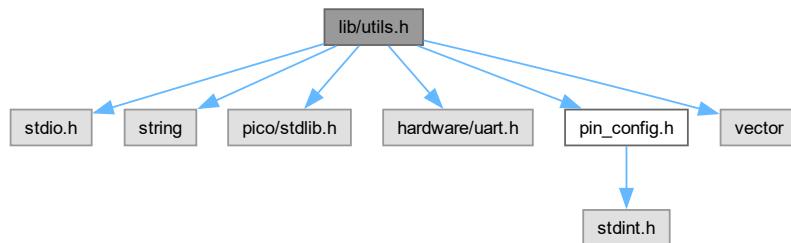
```

8.109 lib/utils.h File Reference

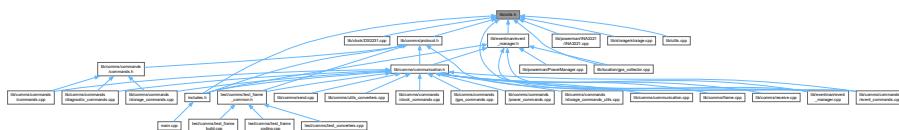
Utility functions and definitions for the Kabisat firmware.

```
#include <stdio.h>
#include <string>
#include "pico/stl.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.
- uint16_t `crc16` (const uint8_t *data, size_t length)
Calculates CRC16 checksum.

Variables

- [VerbosityLevel g_uart_verbosity](#)

Global verbosity level setting for UART output.

8.109.1 Detailed Description

Utility functions and definitions for the Kabisat firmware.

Contains UART logging, color definitions, and CRC calculations

Definition in file [utils.h](#).

8.109.2 Macro Definition Documentation

8.109.2.1 ANSI_RED

```
#define ANSI_RED "\033[31m"
```

ANSI escape codes for terminal color output.

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

8.109.2.2 ANSI_GREEN

```
#define ANSI_GREEN "\033[32m"
```

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

8.109.2.3 ANSI_YELLOW

```
#define ANSI_YELLOW "\033[33m"
```

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

8.109.2.4 ANSI_BLUE

```
#define ANSI_BLUE "\033[34m"
```

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

8.109.2.5 ANSI_CYAN

```
#define ANSI_CYAN "\033[36m"
```

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

8.109.2.6 ANSI_RESET

```
#define ANSI_RESET "\033[0m"
```

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

8.109.3 Enumeration Type Documentation

8.109.3.1 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](#).

8.109.4 Function Documentation

8.109.4.1 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.

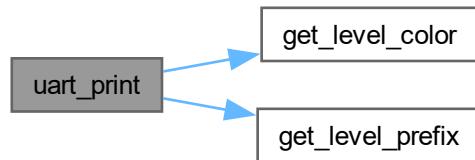
Parameters

<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.

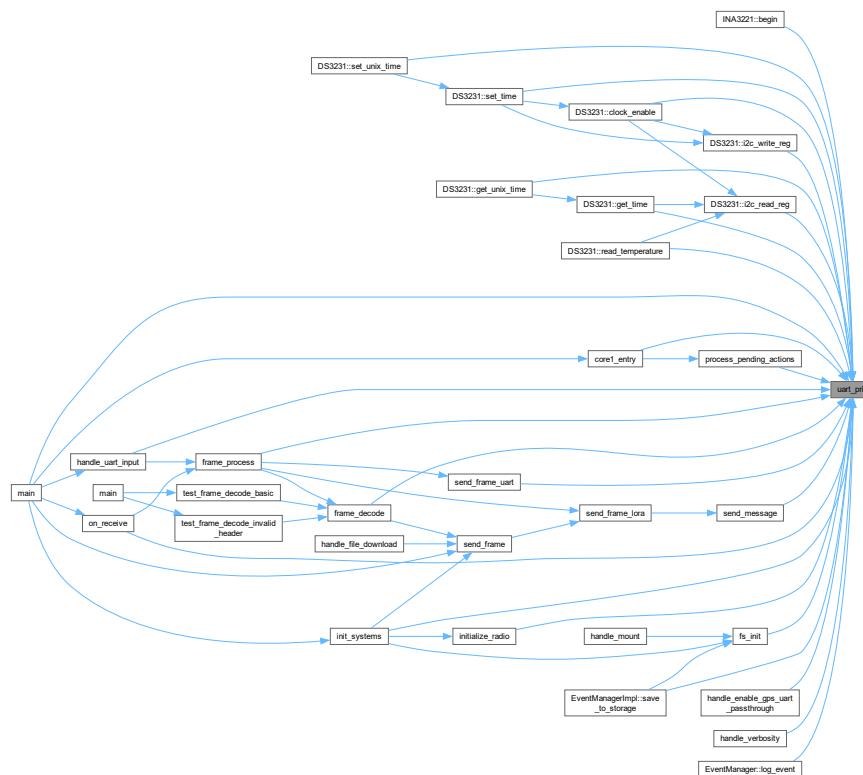
Prints the given message to the specified UART, prepending it with a timestamp and the core number. Uses a mutex to ensure thread-safe access to the UART.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.109.4.2 crc16()

```

uint16_t crc16 (
    const uint8_t * data,
    size_t length)
  
```

Calculates CRC16 checksum.

Parameters

<i>data</i>	Pointer to data buffer
<i>length</i>	Length of data in bytes

Returns

Calculated CRC16 value

Calculates CRC16 checksum.

Parameters

<i>data</i>	A pointer to the data buffer.
<i>length</i>	The length of the data in bytes.

Returns

The CRC16 checksum.

Calculates the CRC16 checksum using the standard algorithm.

Definition at line 97 of file [utils.cpp](#).

8.109.5 Variable Documentation

8.109.5.1 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 18 of file [utils.cpp](#).

8.110 utils.h

[Go to the documentation of this file.](#)

```

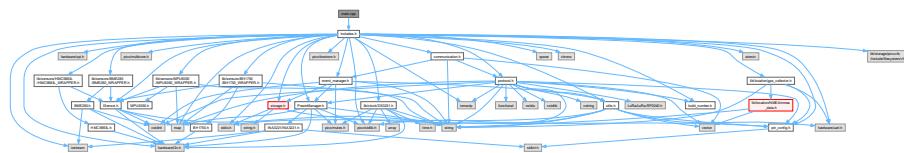
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003
00004 #include <stdio.h>
00005 #include <string>
00006 #include "pico/stl.h"
00007 #include "hardware/uart.h"
00008 #include "pin_config.h"
00009 #include <vector>
00010
00011
00012
00013
00014
00015
00016
00017
00018
00019 #define ANSI_RED      "\033[31m"
00020 #define ANSI_GREEN    "\033[32m"
00021 #define ANSI_YELLOW   "\033[33m"
00022 #define ANSI_BLUE     "\033[34m"
00023 #define ANSI_CYAN     "\033[36m"
00024 #define ANSI_RESET    "\033[0m"
00025
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
00041 extern VerbosityLevel g_uart_verbosity;
00042
00043
00044 void uart_print(const std::string& msg,
00045                  VerbosityLevel level = VerbosityLevel::INFO,
00046                  bool logToFile = false,
00047                  uart_inst_t* uart = DEBUG_UART_PORT);
00048
00049
00050
00051
00052 uint16_t crc16(const uint8_t *data, size_t length);
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076
00077
00078 #endif

```

8.111 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
- char buffer [BUFFER_SIZE]
- int bufferIndex = 0

8.111.1 Macro Definition Documentation

8.111.1.1 LOG_FILENAME

```
#define LOG_FILENAME "/log.txt"
```

Definition at line 3 of file [main.cpp](#).

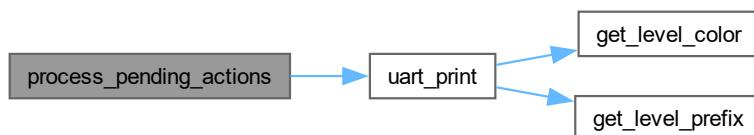
8.111.2 Function Documentation

8.111.2.1 process_pending_actions()

```
void process_pending_actions ()
```

Definition at line 13 of file [main.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

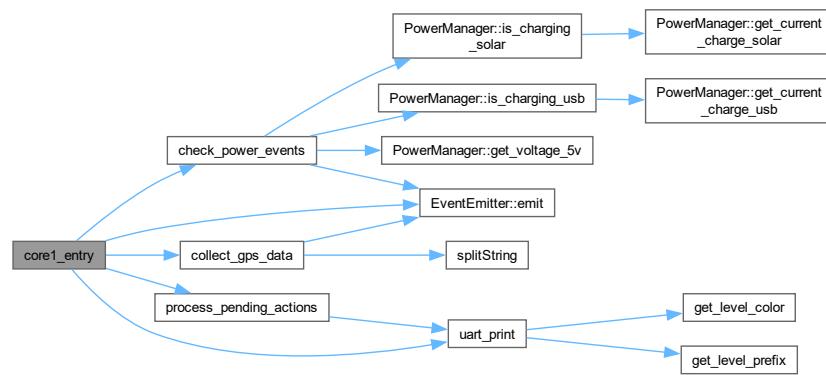


8.111.2.2 core1_entry()

```
void core1_entry ()
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

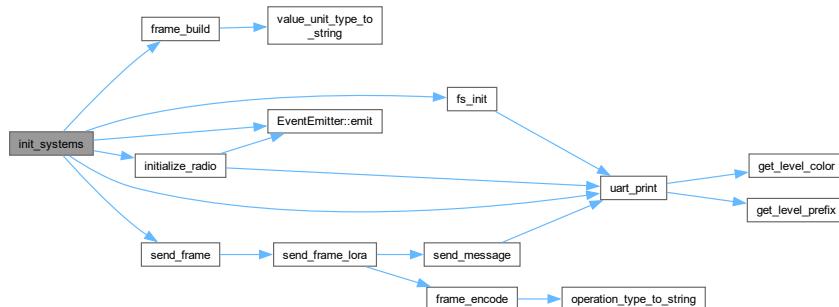


8.111.2.3 init_systems()

```
bool init_systems ()
```

Definition at line 32 of file [main.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

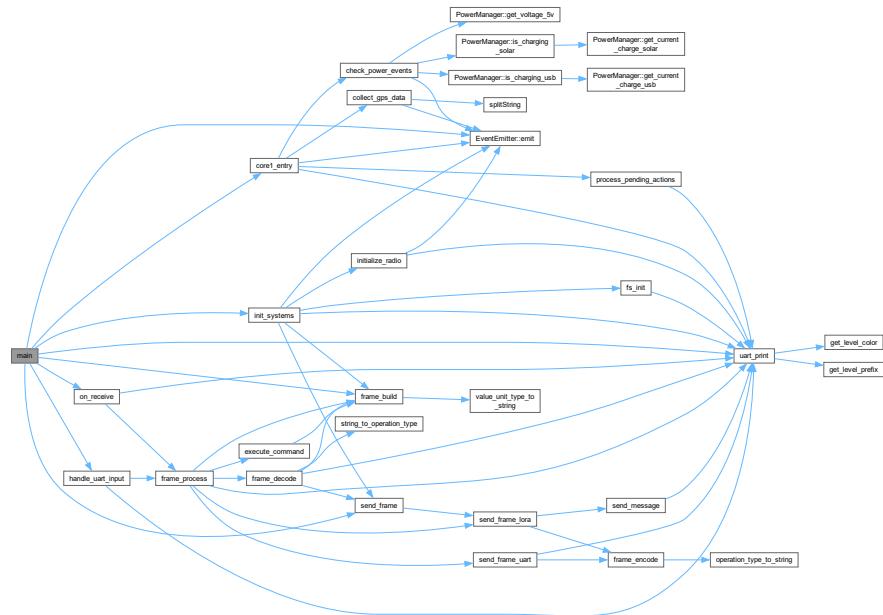


8.111.2.4 main()

```
int main (
    void )
```

Definition at line 108 of file [main.cpp](#).

Here is the call graph for this function:



8.111.3 Variable Documentation

8.111.3.1 powerManager

```
PowerManager powerManager(MAIN_I2C_PORT) (
    MAIN_I2C_PORT )
```

8.111.3.2 systemClock

```
DS3231 systemClock(MAIN_I2C_PORT) (
    MAIN_I2C_PORT )
```

8.111.3.3 buffer

```
char buffer[BUFFER_SIZE]
```

Definition at line 9 of file [main.cpp](#).

8.111.3.4 bufferIndex

```
int bufferIndex = 0
```

Definition at line 10 of file [main.cpp](#).

8.112 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
00009 char buffer[BUFFER_SIZE];
00010 int bufferIndex = 0;
00011
00012
00013 void process_pending_actions() {
00014     if (g_pending_bootloader_reset) {
00015         sleep_ms(100);
00016         uart_print("Entering BOOTSEL mode...", VerbosityLevel::WARNING);
00017         reset_usb_boot(0, 0);
00018     }
00019 }
00020
00021 void core1_entry() {
00022     uart_print("Starting core 1", VerbosityLevel::DEBUG);
00023     EventEmitter::emit(EventGroup::SYSTEM, SystemEvent::CORE1_START);
00024     while (true) {
00025         collect_gps_data();
00026         check_power_events(powerManager);
00027         process_pending_actions();
00028         sleep_ms(10);
00029     }
00030 }
00031
00032 bool init_systems() {
00033     stdio_init_all();
00034
00035     uart_init(DEBUG_UART_PORT, DEBUG_UART_BAUD_RATE);
00036     gpio_set_function(DEBUG_UART_TX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_TX_PIN));
00037     gpio_set_function(DEBUG_UART_RX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_RX_PIN));
00038
00039     uart_init(GPS_UART_PORT, GPS_UART_BAUD_RATE);
00040     gpio_set_function(GPS_UART_TX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_TX_PIN));
00041     gpio_set_function(GPS_UART_RX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_RX_PIN));
00042
00043     gpio_init(PICO_DEFAULT_LED_PIN);
00044     gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
00045
00046     i2c_init(MAIN_I2C_PORT, 400 * 1000);
00047     gpio_set_function(MAIN_I2C_SCL_PIN, GPIO_FUNC_I2C);
```

```

00048     gpio_set_function(MAIN_I2C_SDA_PIN, GPIO_FUNC_I2C);
00049     gpio_pull_up(MAIN_I2C_SCL_PIN);
00050     gpio_pull_up(MAIN_I2C_SDA_PIN);
00051
00052     gpio_init(GPS_POWER_ENABLE_PIN);
00053     gpio_set_dir(GPS_POWER_ENABLE_PIN, GPIO_OUT);
00054     gpio_put(GPS_POWER_ENABLE_PIN, 1);
00055
00056     EventEmitter::emit(EventGroup::GPS, GPSEvent::POWER_ON);
00057
00058     system("color");
00059
00060     bool radioInitSuccess = false;
00061     radioInitSuccess = initialize_radio();
00062
00063     bool sdInitDone = fs_init();
00064     if (sdInitDone) {
00065         FILE *fp = fopen(LOG_FILENAME, "w");
00066         if (fp) {
00067             uart_print("Log file opened.", VerbosityLevel::DEBUG);
00068             int bytesWritten = fprintf(fp, "System init started.\n");
00069             uart_print("Written " + std::to_string(bytesWritten) + " bytes.", VerbosityLevel::DEBUG);
00070             int closeStatus = fclose(fp);
00071             uart_print("Close file status: " + std::to_string(closeStatus), VerbosityLevel::DEBUG);
00072
00073             struct stat file_stat;
00074             if (stat(LOG_FILENAME, &file_stat) == 0) {
00075                 size_t fileSize = file_stat.st_size;
00076                 uart_print("File size: " + std::to_string(fileSize) + " bytes",
00077                           VerbosityLevel::DEBUG);
00077             } else {
00078                 uart_print("Failed to get file size", VerbosityLevel::ERROR);
00079             }
00080
00081             uart_print("File path: /" + std::string(LOG_FILENAME), VerbosityLevel::DEBUG);
00082         } else {
00083             uart_print("Failed to open log file for writing.", VerbosityLevel::ERROR);
00084         }
00085     }
00086
00087     if (sdInitDone) {
00088         uart_print("SD card init: OK", VerbosityLevel::DEBUG);
00089     } else {
00090         uart_print("SD card init: FAILED", VerbosityLevel::ERROR);
00091     }
00092
00093     if (radioInitSuccess) {
00094         uart_print("Radio init: OK", VerbosityLevel::DEBUG);
00095         EventEmitter::emit(EventGroup::COMMS, CommsEvent::RADIO_INIT);
00096     } else {
00097         uart_print("Radio init: FAILED", VerbosityLevel::ERROR);
00098         EventEmitter::emit(EventGroup::COMMS, CommsEvent::RADIO_ERROR);
00099     }
00100
00101     Frame boot = frame_build(OperationType::RES, 0, 0, "HELLO");
00102     send_frame(boot);
00103
00104     return radioInitSuccess;
00105 }
00106
00107
00108 int main()
00109 {
00110     init_systems();
00111     EventEmitter::emit(EventGroup::SYSTEM, SystemEvent::BOOT);
00112     multicore_launch_core1(core1_entry);
00113
00114     gpio_put(PICO_DEFAULT_LED_PIN, 0);
00115
00116     bool powerManagerInitStatus = powerManager.initialize();
00117     if (powerManagerInitStatus)
00118     {
00119         std::map<std::string, std::string> powerConfig = {
00120             {"operating_mode", "continuous"},
00121             {"averaging_mode", "16"},
00122         };
00123         powerManager.configure(powerConfig);
00124     } else {
00125         uart_print("Power manager init error", VerbosityLevel::ERROR);
00126     }
00127
00128     Frame boot = frame_build(OperationType::RES, 0, 0, "START");
00129     send_frame(boot);
00130
00131     std::string bootString = "System init completed @ " +
00132     std::to_string(to_ms_since_boot(get_absolute_time())) + " ms";
00132     uart_print(bootString, VerbosityLevel::WARNING);

```

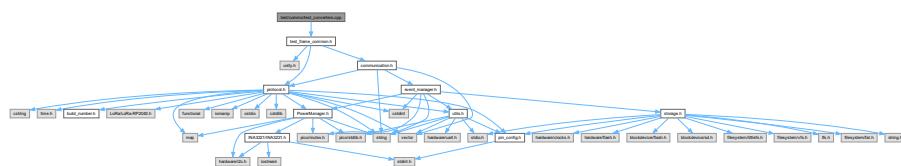
```

00133     gpio_put(PICO_DEFAULT_LED_PIN, 1);
00134
00135     while (true)
00136     {
00137         int packetSize = LoRa.parse_packet();
00138         if (packetSize)
00139         {
00140             on_receive(packetSize);
00141         }
00142
00143         handle_uart_input();
00144     }
00145
00146     return 0;
00147
00148 }
```

8.113 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 \(\)](#)

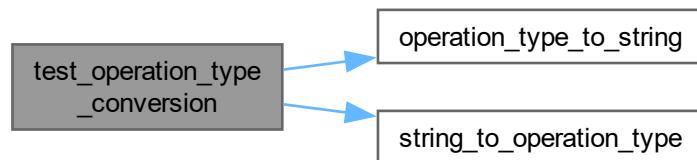
8.113.1 Function Documentation

8.113.1.1 [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:



8.113.1.2 test_value_unit_type_conversion()

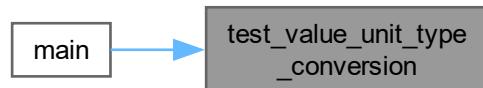
```
void test_value_unit_type_conversion (
    void )
```

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:

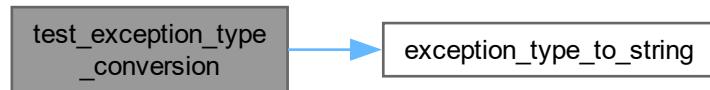


8.113.1.3 test_exception_type_conversion()

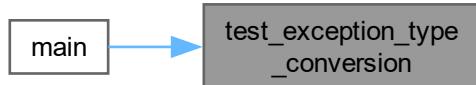
```
void test_exception_type_conversion (
    void )
```

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:



8.113.1.4 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:



8.114 test_converters.cpp

[Go to the documentation of this file.](#)

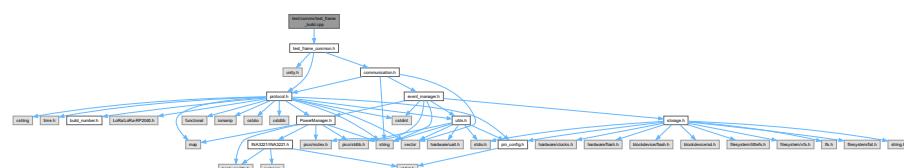
```

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
00013 void test_value_unit_type_conversion() {
00014     ValueUnit unit = ValueUnit::VOLT;
00015     std::string str = value_unit_type_to_string(unit);
00016
00017     TEST_ASSERT_EQUAL_STRING("V", str.c_str());
00018 }
00019
00020 void test_exception_type_conversion() {
00021     ExceptionType type = ExceptionType::INVALID_PARAM;
00022     std::string str = exception_type_to_string(type);
00023
00024     TEST_ASSERT_EQUAL_STRING("INVALID PARAM", str.c_str());
00025 }
00026
00027 void test_hex_string_conversion() {
00028     std::string hex = "0A0B0C";
00029     std::vector<uint8_t> bytes = hex_string_to_bytes(hex);
00030
00031     TEST_ASSERT_EQUAL(3, bytes.size());
00032     TEST_ASSERT_EQUAL(0x0A, bytes[0]);
00033     TEST_ASSERT_EQUAL(0x0B, bytes[1]);
00034     TEST_ASSERT_EQUAL(0x0C, bytes[2]);
00035 }
  
```

8.115 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 ()`

8.115.1 Function Documentation

8.115.1.1 `test_frame_build_success()`

```
void test_frame_build_success (
    void )
```

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:

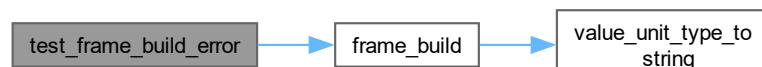


8.115.1.2 `test_frame_build_error()`

```
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:



8.115.1.3 test_frame_build_info()

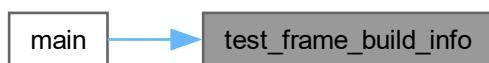
```
void test_frame_build_info (
    void )
```

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:



8.116 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);
```

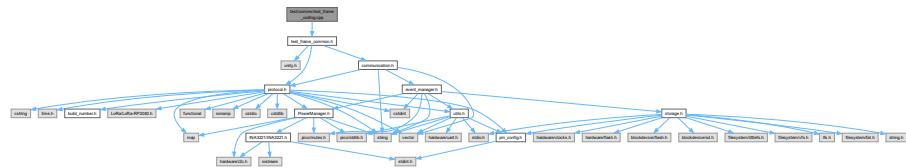
```

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 }

```

8.117 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\(\)](#)

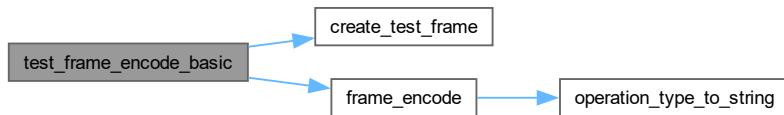
8.117.1 Function Documentation

8.117.1.1 [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:

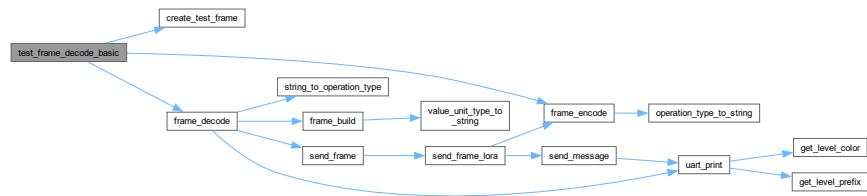


8.117.1.2 test_frame_decode_basic()

```
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:

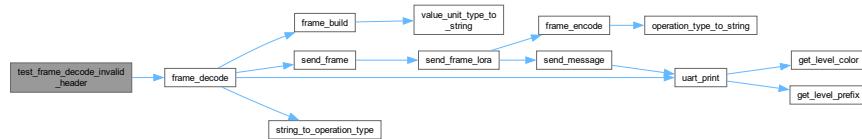


8.117.1.3 test_frame_decode_invalid_header()

```
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:



8.118 test_frame_coding.cpp

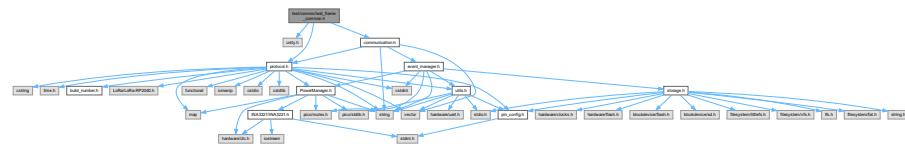
[Go to the documentation of this file.](#)

```

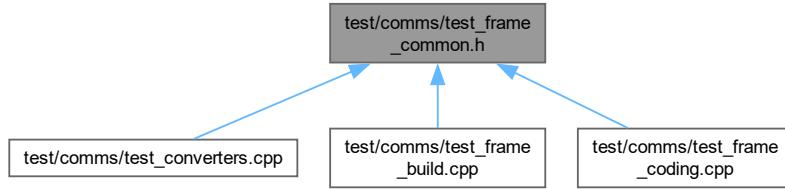
00001 // test_frame_codec.cpp
00002 #include "test_frame_common.h"
00003
00004 void test_frame_encode_basic() {
00005     Frame frame = create_test_frame();
00006     std::string encoded = frame_encode(frame);
00007
00008     TEST_ASSERT_NOT_EQUAL(0, encoded.length());
00009     TEST_ASSERT_TRUE(encoded.find(FRAME_BEGIN) != 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 }
```

8.119 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 \(\)](#)

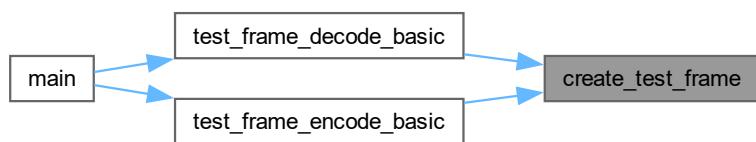
8.119.1 Function Documentation

8.119.1.1 [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:



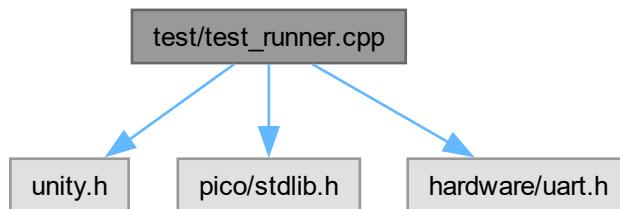
8.120 test_frame_common.h

[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
0010 Frame create_test_frame() {
0011     Frame frame;
0012     frame.header = FRAME_BEGIN;
0013     frame.direction = 1;
0014     frame.operationType = OperationType::GET;
0015     frame.group = 1;
0016     frame.command = 2;
0017     frame.value = "test_value";
0018     frame.unit = "V";
0019     frame.footer = FRAME_END;
0020     return frame;
0021 }
0022
0023 #endif
```

8.121 test/test_runner.cpp File Reference

```
#include "unity.h"
#include "pico/stdlib.h"
#include "hardware/uart.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)
- int `main` (void)

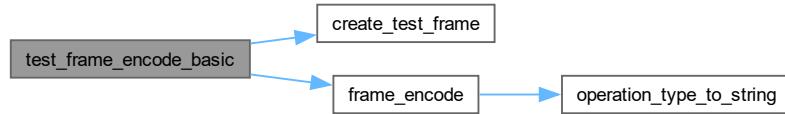
8.121.1 Function Documentation

8.121.1.1 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:

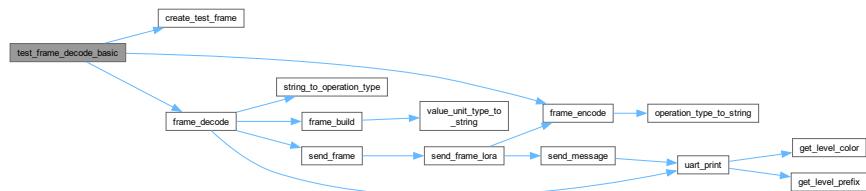


8.121.1.2 test_frame_decode_basic()

```
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:

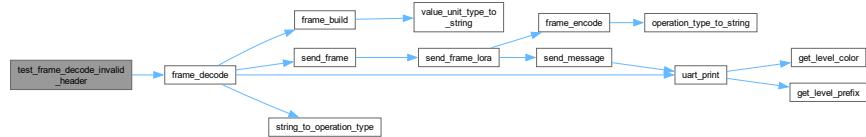


8.121.1.3 test_frame_decode_invalid_header()

```
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:



8.121.1.4 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:



8.121.1.5 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:



8.121.1.6 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:

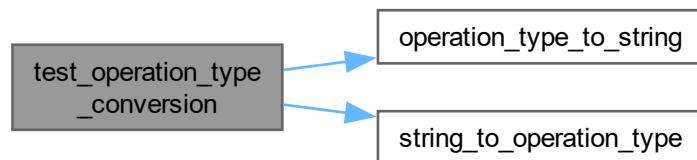


8.121.1.7 test_operation_type_conversion()

```
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:



8.121.1.8 test_value_unit_type_conversion()

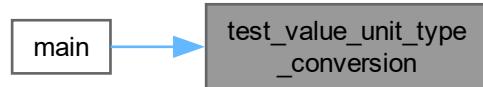
```
void test_value_unit_type_conversion (
    void ) [extern]
```

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:

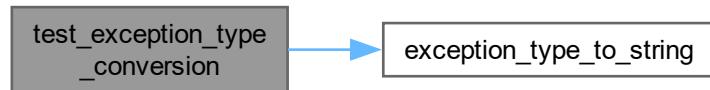


8.121.1.9 test_exception_type_conversion()

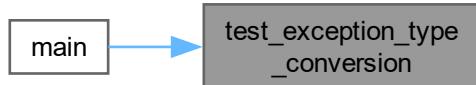
```
void test_exception_type_conversion (
    void ) [extern]
```

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:



8.121.1.10 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:

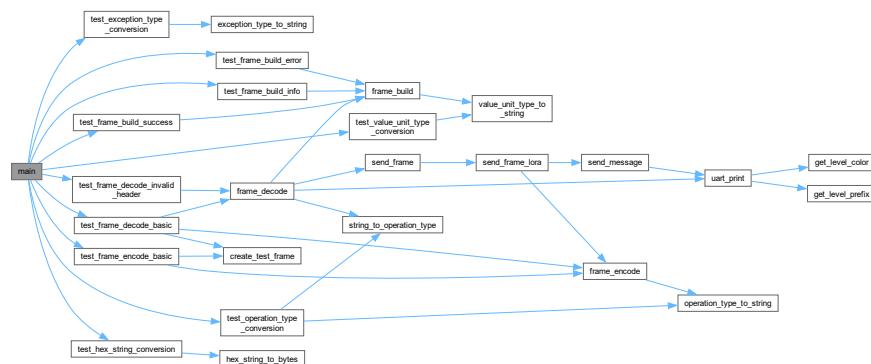


8.121.1.11 main()

```
int main (
    void )
```

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

Here is the call graph for this function:



8.122 test_runner.cpp

[Go to the documentation of this file.](#)

```

00001 // test_runner.cpp
00002 #include "unity.h"
00003 #include "pico/stl.h"
00004 #include "hardware/uart.h"
00005
00006 // External test function declarations
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 int main(void) {
00019     stdio_init_all();
00020     uart_init(uart0, 115200);
  
```

```
00021     gpio_set_function(0, GPIO_FUNC_UART);
00022     gpio_set_function(1, GPIO_FUNC_UART);
00023
00024     UNITY_BEGIN();
00025     uart_puts(uart0, "begin unity tests\n");
00026
00027     // Frame codec tests
00028     uart_puts(uart0, "begin frame codec tests\n");
00029     RUN_TEST(test_frame_encode_basic);
00030     RUN_TEST(test_frame_decode_basic);
00031     RUN_TEST(test_frame_decode_invalid_header);
00032     uart_puts(uart0, "end frame codec tests\n");
00033
00034     // Frame build tests
00035     uart_puts(uart0, "begin frame build tests\n");
00036     RUN_TEST(test_frame_build_success);
00037     RUN_TEST(test_frame_build_error);
00038     RUN_TEST(test_frame_build_info);
00039     uart_puts(uart0, "end frame build tests\n");
00040
00041     // Converter tests
00042     uart_puts(uart0, "begin converter tests\n");
00043     RUN_TEST(test_operation_type_conversion);
00044     RUN_TEST(test_value_unit_type_conversion);
00045     RUN_TEST(test_exception_type_conversion);
00046     RUN_TEST(test_hex_string_conversion);
00047     uart_puts(uart0, "end converter tests\n");
00048
00049     return UNITY_END();
00050 }
```

Index

_BH1750_DEFAULT_MTREG
 BH1750.h, [256](#)
_BH1750_DEVICE_ID
 BH1750.h, [255](#)
_BH1750_MTREG_MAX
 BH1750.h, [255](#)
_BH1750_MTREG_MIN
 BH1750.h, [255](#)
__attribute__
 Event Manager, [46](#), [49](#)
_filterRes
 INA3221, [126](#)
_i2c
 INA3221, [126](#)
_i2c_addr
 BH1750, [72](#)
 INA3221, [126](#)
_masken_reg
 INA3221, [126](#)
_read
 INA3221, [122](#)
_shuntRes
 INA3221, [126](#)
_write
 INA3221, [123](#)
~EventManager
 EventManager, [106](#)
~ISensor
 ISensor, [127](#)

ACCEL_X
 ISensor.h, [276](#)
ACCEL_Y
 ISensor.h, [276](#)
ACCEL_Z
 ISensor.h, [276](#)
ADDR_SDO_HIGH
 BME280, [78](#)
ADDR_SDO_LOW
 BME280, [78](#)
address
 HMC5883L, [117](#)
Alert Functions, [62](#)
 enable_alerts, [64](#)
 get_crit_alert, [65](#)
 get_power_valid_alert, [67](#)
 get_warn_alert, [65](#)
 set_alert_latch, [67](#)
 set_crit_alert_limit, [63](#)
 set_power_valid_limit, [64](#)

 set_warn_alert_limit, [63](#)
ANSI_BLUE
 utils.h, [294](#)
ANSI_CYAN
 utils.h, [294](#)
ANSI_GREEN
 utils.h, [294](#)
ANSI_RED
 utils.h, [294](#)
ANSI_RESET
 utils.h, [294](#)
ANSI_YELLOW
 utils.h, [294](#)
avg_mode
 INA3221::conf_reg_t, [91](#)

bcd_to_bin
 DS3231, [98](#)

begin
 BH1750, [70](#)
 Configuration Functions, [52](#)

BH1750, [69](#)
 _i2c_addr, [72](#)
 begin, [70](#)
 BH1750, [70](#)
 configure, [70](#)
 CONTINUOUS_HIGH_RES_MODE, [70](#)
 CONTINUOUS_HIGH_RES_MODE_2, [70](#)
 CONTINUOUS_LOW_RES_MODE, [70](#)
 get_light_level, [71](#)
 Mode, [69](#)
 ONE_TIME_HIGH_RES_MODE, [70](#)
 ONE_TIME_HIGH_RES_MODE_2, [70](#)
 ONE_TIME_LOW_RES_MODE, [70](#)
 POWER_ON, [70](#)
 RESET, [70](#)
 UNCONFIGURED_POWER_DOWN, [70](#)
 write8, [71](#)

BH1750.h
 _BH1750_DEFAULT_MTREG, [256](#)
 _BH1750_DEVICE_ID, [255](#)
 _BH1750_MTREG_MAX, [255](#)
 _BH1750_MTREG_MIN, [255](#)

BH1750Wrapper, [72](#)
 BH1750Wrapper, [73](#)
 configure, [74](#)
 get_i2c_addr, [73](#)
 get_type, [74](#)
 init, [73](#)
 initialized_, [74](#)

is_initialized, 74
 read_data, 73
 sensor_, 74
 bin_to_bcd
 DS3231, 97
 BME280, 75
 ADDR_SDO_HIGH, 78
 ADDR_SDO_LOW, 78
 BME280, 76
 calib_params, 79
 configure_sensor, 78
 convert_humidity, 77
 convert_pressure, 77
 convert_temperature, 77
 device_addr, 79
 get_calibration_parameters, 78
 i2c_port, 79
 init, 77
 initialized_, 79
 NUM_CALIB_PARAMS, 84
 read_raw_all, 77
 REG_CONFIG, 79
 REG_CTRL_HUM, 79
 REG_CTRL_MEAS, 79
 REG_DIG_H1, 83
 REG_DIG_H2, 83
 REG_DIG_H3, 83
 REG_DIG_H4, 83
 REG_DIG_H5, 84
 REG_DIG_H6, 84
 REG_DIG_P1_LSB, 81
 REG_DIG_P1_MSB, 81
 REG_DIG_P2_LSB, 81
 REG_DIG_P2_MSB, 81
 REG_DIG_P3_LSB, 81
 REG_DIG_P3_MSB, 81
 REG_DIG_P4_LSB, 82
 REG_DIG_P4_MSB, 82
 REG_DIG_P5_LSB, 82
 REG_DIG_P5_MSB, 82
 REG_DIG_P6_LSB, 82
 REG_DIG_P6_MSB, 82
 REG_DIG_P7_LSB, 82
 REG_DIG_P7_MSB, 82
 REG_DIG_P8_LSB, 83
 REG_DIG_P8_MSB, 83
 REG_DIG_P9_LSB, 83
 REG_DIG_P9_MSB, 83
 REG_DIG_T1_LSB, 80
 REG_DIG_T1_MSB, 80
 REG_DIG_T2_LSB, 80
 REG_DIG_T2_MSB, 80
 REG_DIG_T3_LSB, 81
 REG_DIG_T3_MSB, 81
 REG_HUMIDITY_MSB, 80
 REG_PRESSURE_MSB, 80
 REG_RESET, 80
 REG_TEMPERATURE_MSB, 80
 reset, 77
 t_fine, 79
 BME280CalibParam, 84
 dig_h1, 86
 dig_h2, 86
 dig_h3, 86
 dig_h4, 87
 dig_h5, 87
 dig_h6, 87
 dig_p1, 85
 dig_p2, 85
 dig_p3, 85
 dig_p4, 85
 dig_p5, 86
 dig_p6, 86
 dig_p7, 86
 dig_p8, 86
 dig_p9, 86
 dig_t1, 85
 dig_t2, 85
 dig_t3, 85
 BME280Wrapper, 87
 BME280Wrapper, 88
 configure, 89
 get_type, 89
 init, 89
 initialized_, 89
 is_initialized, 89
 read_data, 89
 sensor_, 89
 BOOL
 protocol.h, 193
 BOOT
 Event Manager, 44
 buffer
 main.cpp, 302
 BUFFER_SIZE
 pin_config.h, 233
 bufferIndex
 main.cpp, 302
 BUILD_NUMBER
 build_number.h, 145
 build_number.h, 145
 BUILD_NUMBER, 145
 bus_conv_time
 INA3221::conf_reg_t, 91
 calculate_checksum
 storage_commands_utils.cpp, 176
 storage_commands_utils.h, 178
 calib_params
 BME280, 79
 century
 ds3231_data_t, 100
 ch1_en
 INA3221::conf_reg_t, 91
 ch2_en
 INA3221::conf_reg_t, 91
 ch3_en

INA3221::conf_reg_t, 91
CHANGED
 Event Manager, 45
charging_solar_active_
 PowerManager, 141
charging_usb_active_
 PowerManager, 141
check_power_alerts
 PowerManager, 139
check_power_events
 Event Manager, 45
CLOCK
 Event Manager, 44
Clock Commands, 1
Clock Management Commands, 11
 handle_clock_sync_interval, 13
 handle_get_last_sync_time, 13
 handle_time, 11
 handle_timezone_offset, 12
 systemClock, 14
clock_commands.cpp
 CLOCK_GROUP, 155
 CLOCK_SYNC_INTERVAL, 156
 LAST_SYNC_TIME, 156
 TIME, 155
 TIMEZONE_OFFSET, 155
clock_enable
 DS3231, 95
CLOCK_GROUP
 clock_commands.cpp, 155
clock_mutex_
 DS3231, 99
CLOCK_SYNC_INTERVAL
 clock_commands.cpp, 156
ClockEvent
 Event Manager, 45
collect_gps_data
 gps_collector.cpp, 222
 gps_collector.h, 224
command
 Frame, 114
Command System, 14
 CommandHandler, 15
 commandHandlers, 16
 CommandMap, 15
 execute_command, 15
CommandAccessLevel
 protocol.h, 192
CommandHandler
 Command System, 15
 frame.cpp, 189
commandHandlers
 Command System, 16
CommandMap
 Command System, 15
COMMS
 Event Manager, 44
CommsEvent

Event Manager, 44
communication.cpp
 initialize_radio, 179
 interval, 181
 lastPrintTime, 181
 lastReceiveTime, 181
 lastSendTime, 180
 msgCount, 180
 outgoing, 180
communication.h
 determine_unit, 187
 handle_uart_input, 184
 initialize_radio, 183
 on_receive, 183
 send_frame, 185
 send_frame_lora, 186
 send_frame_uart, 185
 send_message, 185
 split_and_send_message, 187
Configuration Functions, 50
 begin, 52
 get_die_id, 53
 get_manufacturer_id, 53
 INA3221, 51
 read_register, 54
 reset, 52
 set_averaging_mode, 58
 set_bus_conversion_time, 59
 set_bus_measurement_disable, 58
 set_bus_measurement_enable, 57
 set_mode_continuous, 55
 set_mode_power_down, 55
 set_mode_triggered, 56
 set_shunt_conversion_time, 59
 set_shunt_measurement_disable, 57
 set_shunt_measurement_enable, 56
configure
 BH1750, 70
 BH1750Wrapper, 74
 BME280Wrapper, 89
 HMC5883LWrapper, 120
 ISensor, 127
 MPU6050Wrapper, 132
 PowerManager, 138
configure_sensor
 BME280, 78
CONTINUOUS_HIGH_RES_MODE
 BH1750, 70
CONTINUOUS_HIGH_RES_MODE_2
 BH1750, 70
CONTINUOUS_LOW_RES_MODE
 BH1750, 70
conv_ready
 INA3221::masken_reg_t, 128
convert_humidity
 BME280, 77
convert_pressure
 BME280, 77

convert_temperature
 BME280, 77
 core1_entry
 main.cpp, 299
 CORE1_START
 Event Manager, 44
 CORE1_STOP
 Event Manager, 44
 crc16
 utils.cpp, 290
 utils.h, 296
 create_test_frame
 test_frame_common.h, 313
 crit_alert_ch1
 INA3221::masken_reg_t, 129
 crit_alert_ch2
 INA3221::masken_reg_t, 129
 crit_alert_ch3
 INA3221::masken_reg_t, 129
 crit_alert_latch_en
 INA3221::masken_reg_t, 130

 DATA_COMMAND
 storage_commands.cpp, 172

 DATA_READY
 Event Manager, 45

 date
 ds3231_data_t, 100

 DATETIME
 protocol.h, 193

 day
 ds3231_data_t, 100

 days_of_week
 DS3231.h, 153

 DEBUG
 utils.h, 295

 DEBUG_UART_BAUD_RATE
 pin_config.h, 232

 DEBUG_UART_PORT
 pin_config.h, 232

 DEBUG_UART_RX_PIN
 pin_config.h, 232

 DEBUG_UART_TX_PIN
 pin_config.h, 232

 DELIMITER
 protocol.h, 198

 determine_unit
 communication.h, 187

 device_addr
 BME280, 79

 Diagnostic Commands, 17
 g_pending_bootloader_reset, 21
 handle_enter_bootloader_mode, 20
 handle_get_build_version, 18
 handle_get_commands_list, 17
 handle_verbosity, 19

 dig_h1
 BME280CalibParam, 86

 dig_h2
 BME280CalibParam, 86

 dig_h3
 BME280CalibParam, 86

 dig_h4
 BME280CalibParam, 87

 dig_h5
 BME280CalibParam, 87

 dig_h6
 BME280CalibParam, 87

 dig_p1
 BME280CalibParam, 85

 dig_p2
 BME280CalibParam, 85

 dig_p3
 BME280CalibParam, 85

 dig_p4
 BME280CalibParam, 85

 dig_p5
 BME280CalibParam, 86

 dig_p6
 BME280CalibParam, 86

 dig_p7
 BME280CalibParam, 86

 dig_p8
 BME280CalibParam, 86

 dig_p9
 BME280CalibParam, 86

 dig_t1
 BME280CalibParam, 85

 dig_t2
 BME280CalibParam, 85

 dig_t3
 BME280CalibParam, 85

 direction
 Frame, 114

 DS3231, 92
 bcd_to_bin, 98
 bin_to_bcd, 97
 clock_enable, 95
 clock_mutex_, 99
 DS3231, 93
 ds3231_addr, 99
 get_time, 93
 get_unix_time, 95
 i2c, 99
 i2c_read_reg, 96
 i2c_write_reg, 96
 read_temperature, 94
 set_time, 93
 set_unix_time, 94

 DS3231.h
 days_of_week, 153
 DS3231_CONTROL_REG, 153
 DS3231_CONTROL_STATUS_REG, 153
 DS3231_DATE_REG, 152
 DS3231_DAY_REG, 152
 DS3231_DEVICE_ADRESS, 152
 DS3231_HOURS_REG, 152

DS3231_MINUTES_REG, 152
DS3231_MONTH_REG, 152
DS3231_SECONDS_REG, 152
DS3231_TEMPERATURE_LSB_REG, 153
DS3231_TEMPERATURE_MSB_REG, 153
DS3231_YEAR_REG, 153
FRIDAY, 153
MONDAY, 153
SATURDAY, 153
SUNDAY, 153
THURSDAY, 153
TUESDAY, 153
WEDNESDAY, 153
ds3231_addr
 DS3231, 99
DS3231_CONTROL_REG
 DS3231.h, 153
DS3231_CONTROL_STATUS_REG
 DS3231.h, 153
ds3231_data_t, 99
 century, 100
 date, 100
 day, 100
 hours, 100
 minutes, 100
 month, 100
 seconds, 100
 year, 100
DS3231_DATE_REG
 DS3231.h, 152
DS3231_DAY_REG
 DS3231.h, 152
DS3231_DEVICE_ADDRESS
 DS3231.h, 152
DS3231_HOURS_REG
 DS3231.h, 152
DS3231_MINUTES_REG
 DS3231.h, 152
DS3231_MONTH_REG
 DS3231.h, 152
DS3231_SECONDS_REG
 DS3231.h, 152
DS3231_TEMPERATURE_LSB_REG
 DS3231.h, 153
DS3231_TEMPERATURE_MSB_REG
 DS3231.h, 153
DS3231_YEAR_REG
 DS3231.h, 153

emit
 EventEmitter, 101
enable_alerts
 Alert Functions, 64
END_COMMAND
 storage_commands.cpp, 172
ENVIRONMENT
 ISensor.h, 276
ERR
 protocol.h, 192

ERROR
 Event Manager, 45
 utils.h, 295
EVENT
 utils.h, 295
event
 event_manager.h, 218
 EventLog, 104
Event Commands, 21
 handle_get_event_count, 22
 handle_get_last_events, 21
Event Manager, 41
 __attribute__, 46, 49
 BOOT, 44
 CHANGED, 45
 check_power_events, 45
 CLOCK, 44
 ClockEvent, 45
 COMMS, 44
 CommsEvent, 44
 CORE1_START, 44
 CORE1_STOP, 44
 DATA_READY, 45
 ERROR, 45
 EventGroup, 43
 eventId, 48
 eventManager, 49
 FALL_RATE_THRESHOLD, 48
 FALLING_TREND_REQUIRED, 48
 fallingTrendCount, 48
 get_event, 47
 GPS, 44
 GPS_SYNC, 45
 GPSEvent, 45
 lastPowerState, 48
 lastSolarState, 49
 lastUSBState, 49
 LOCK, 45
 log_event, 46
 LOST, 45
 LOW_BATTERY, 44
 MSG_RECEIVED, 45
 MSG_SENT, 45
 OVERCHARGE, 44
 PASS_THROUGH_END, 45
 PASS_THROUGH_START, 45
 POWER, 44
 POWER_FALLING, 44
 POWER_NORMAL, 44
 POWER_OFF, 45
 POWER_ON, 45
 PowerEvent, 44
 RADIO_ERROR, 45
 RADIO_INIT, 45
 SHUTDOWN, 44
 SOLAR_ACTIVE, 44
 SOLAR_INACTIVE, 44
 SYSTEM, 44

systemClock, 49
 SystemEvent, 44
 UART_ERROR, 45
 USB_CONNECTED, 44
 USB_DISCONNECTED, 44
 VOLTAGE_LOW_THRESHOLD, 48
 VOLTAGE_OVERCHARGE_THRESHOLD, 48
 WATCHDOG_RESET, 44
EVENT_BUFFER_SIZE
 event_manager.h, 217
EVENT_LOG_FILE
 event_manager.h, 217
event_manager.h
 event, 218
EVENT_BUFFER_SIZE, 217
EVENT_LOG_FILE, 217
 group, 218
 id, 218
 timestamp, 218
 to_string, 217
eventCount
 EventManager, 108
EventEmitter, 101
 emit, 101
EventGroup
 Event Manager, 43
EventLog, 102
 event, 104
 group, 104
 id, 103
 timestamp, 103
 to_string, 103
eventLogId
 Event Manager, 48
EventManager, 104
 ~EventManager, 106
 eventCount, 108
 EventManager, 106
 eventMutex, 108
 events, 108
 get_event_count, 106
 init, 106
 load_from_storage, 107
 needsPersistence, 108
 nextEventId, 108
 save_to_storage, 107
 writeIndex, 108
eventManager
 Event Manager, 49
EventManagerImpl, 109
 EventManagerImpl, 110
 load_from_storage, 111
 save_to_storage, 111
eventMutex
 EventManager, 108
eventRegister
 frame.cpp, 189
events
 EventManager, 108
ExceptionType
 protocol.h, 193
execute_command
 Command System, 15
FALL_RATE_THRESHOLD
 Event Manager, 48
 PowerManager, 140
FALLING_TREND_REQUIRED
 Event Manager, 48
 PowerManager, 140
fallingTrendCount
 Event Manager, 48
fd
 FileHandle, 112
FileHandle, 112
 fd, 112
 is_open, 112
footer
 Frame, 114
Frame, 112
 command, 114
 direction, 114
 footer, 114
 group, 114
 header, 114
 operationType, 114
 unit, 114
 value, 114
Frame Handling, 36
 frame_build, 39
 frame_decode, 37
 frame_encode, 36
 frame_process, 38
frame.cpp
 CommandHandler, 189
 eventRegister, 189
FRAME_BEGIN
 protocol.h, 197
frame_build
 Frame Handling, 39
frame_decode
 Frame Handling, 37
frame_encode
 Frame Handling, 36
FRAME_END
 protocol.h, 197
frame_process
 Frame Handling, 38
FRIDAY
 DS3231.h, 153
fs_close_file
 storage.h, 285
fs_file_exists
 storage.h, 285

fs_init
 storage.cpp, 281
 storage.h, 283
fs_open_file
 storage.h, 284
fs_read_file
 storage.h, 284
fs_write_file
 storage.h, 284

g_pending_bootloader_reset
 Diagnostic Commands, 21

g_uart_verbosity
 utils.cpp, 291
 utils.h, 297

GET
 protocol.h, 192

get_calibration_parameters
 BME280, 78

get_crit_alert
 Alert Functions, 65

get_current
 INA3221, 125

get_current_charge_solar
 PowerManager, 137

get_current_charge_total
 PowerManager, 137

get_current_charge_usb
 PowerManager, 137

get_current_draw
 PowerManager, 137

get_current_ma
 Measurement Functions, 61

get_die_id
 Configuration Functions, 53

get_event
 Event Manager, 47

get_event_count
 EventManager, 106

get_gga_tokens
 NMEAData, 134

get_i2c_addr
 BH1750Wrapper, 73

get_instance
 SensorWrapper, 143

get_level_color
 utils.cpp, 287

get_level_prefix
 utils.cpp, 288

get_light_level
 BH1750, 71

get_manufacturer_id
 Configuration Functions, 53

get_power_valid_alert
 Alert Functions, 67

get_rmc_tokens
 NMEAData, 134

get_shunt_voltage
 Measurement Functions, 60

get_time
 DS3231, 93

get_type
 BH1750Wrapper, 74
 BME280Wrapper, 89
 HMC5883LWrapper, 119
 ISensor, 127
 MPU6050Wrapper, 132

get_unix_time
 DS3231, 95

get_voltage
 Measurement Functions, 62

get_voltage_5v
 PowerManager, 138

get_voltage_battery
 PowerManager, 138

get_warn_alert
 Alert Functions, 65

gga_mutex_
 NMEAData, 135

gga_tokens_
 NMEAData, 134

GPS
 Event Manager, 44

GPS Commands, 23
 handle_enable_gps_uart_passthrough, 24
 handle_get_gga_data, 26
 handle_get_rmc_data, 25
 handle_gps_power_status, 23

gps_collector.cpp
 collect_gps_data, 222
 MAX_RAW_DATA_LENGTH, 221
 nmea_data, 223
 splitString, 222

gps_collector.h
 collect_gps_data, 224

GPS_POWER_ENABLE_PIN
 pin_config.h, 233

GPS_SYNC
 Event Manager, 45

GPS_UART_BAUD_RATE
 pin_config.h, 233

GPS_UART_PORT
 pin_config.h, 233

GPS_UART_RX_PIN
 pin_config.h, 233

GPS_UART_TX_PIN
 pin_config.h, 233

GPSEvent
 Event Manager, 45

group
 event_manager.h, 218
 EventLog, 104
 Frame, 114

GYRO_X
 ISensor.h, 276

GYRO_Y
 ISensor.h, 276

GYRO_Z
ISensor.h, 276

handle_clock_sync_interval
Clock Management Commands, 13

handle_enable_gps_uart_passthrough
GPS Commands, 24

handle_enter_bootloader_mode
Diagnostic Commands, 20

handle_file_download
Storage Commands, 33

handle_get_build_version
Diagnostic Commands, 18

handle_get_commands_list
Diagnostic Commands, 17

handle_get_current_charge_solar
Power Commands, 30

handle_get_current_charge_total
Power Commands, 31

handle_get_current_charge_usb
Power Commands, 30

handle_get_current_draw
Power Commands, 32

handle_get_event_count
Event Commands, 22

handle_get_gga_data
GPS Commands, 26

handle_get_last_events
Event Commands, 21

handle_get_last_sync_time
Clock Management Commands, 13

handle_get_power_manager_ids
Power Commands, 28

handle_get_rmc_data
GPS Commands, 25

handle_get_voltage_5v
Power Commands, 29

handle_get_voltage_battery
Power Commands, 28

handle_gps_power_status
GPS Commands, 23

handle_list_files
Storage Commands, 34

handle_mount
Storage Commands, 35

handle_time
Clock Management Commands, 11

handle_timezone_offset
Clock Management Commands, 12

handle_uart_input
communication.h, 184
receive.cpp, 200

handle_verbosity
Diagnostic Commands, 19

header
Frame, 114

hex_string_to_bytes
protocol.h, 195
utils_converters.cpp, 210

HMC5883L, 115
address, 117
HMC5883L, 115
i2c, 117
init, 116
read, 116
read_register, 117
write_register, 116

HMC5883LWrapper, 118
configure, 120
get_type, 119
HMC5883LWrapper, 119
init, 119
initialized_, 120
is_initialized, 119
read_data, 119
sensor_, 120

hours
ds3231_data_t, 100

HUMIDITY
ISensor.h, 276

i2c
DS3231, 99
HMC5883L, 117

i2c_port
BME280, 79

i2c_read_reg
DS3231, 96

i2c_write_reg
DS3231, 96

id
event_manager.h, 218
EventLog, 103

IMU
ISensor.h, 276

INA3221, 120
_filterRes, 126
_i2c, 126
_i2c_addr, 126
_masken_reg, 126
_read, 122
_shuntRes, 126
_write, 123
Configuration Functions, 51
get_current, 125

INA3221 Power Monitor, 50

INA3221.h
INA3221_ADDR40_GND, 245
INA3221_ADDR41_VCC, 245
INA3221_ADDR42_SDA, 245
INA3221_ADDR43_SCL, 245
ina3221_addr_t, 244
ina3221_avg_mode_t, 246
INA3221_CH1, 245
INA3221_CH2, 245
INA3221_CH3, 245
INA3221_CH_NUM, 246
ina3221_ch_t, 245

ina3221_conv_time_t, 245
INA3221_REG_CH1_BUSV, 245
INA3221_REG_CH1_CRIT_ALERT_LIM, 245
INA3221_REG_CH1_SHUNTV, 245
INA3221_REG_CH1_WARNING_ALERT_LIM,
 245
INA3221_REG_CH2_BUSV, 245
INA3221_REG_CH2_CRIT_ALERT_LIM, 245
INA3221_REG_CH2_SHUNTV, 245
INA3221_REG_CH2_WARNING_ALERT_LIM,
 245
INA3221_REG_CH3_BUSV, 245
INA3221_REG_CH3_CRIT_ALERT_LIM, 245
INA3221_REG_CH3_SHUNTV, 245
INA3221_REG_CH3_WARNING_ALERT_LIM,
 245
INA3221_REG_CONF, 245
INA3221_REG_CONF_AVG_1, 246
INA3221_REG_CONF_AVG_1024, 246
INA3221_REG_CONF_AVG_128, 246
INA3221_REG_CONF_AVG_16, 246
INA3221_REG_CONF_AVG_256, 246
INA3221_REG_CONF_AVG_4, 246
INA3221_REG_CONF_AVG_512, 246
INA3221_REG_CONF_AVG_64, 246
INA3221_REG_CONF_CT_1100US, 246
INA3221_REG_CONF_CT_140US, 246
INA3221_REG_CONF_CT_204US, 246
INA3221_REG_CONF_CT_2116US, 246
INA3221_REG_CONF_CT_332US, 246
INA3221_REG_CONF_CT_4156US, 246
INA3221_REG_CONF_CT_588US, 246
INA3221_REG_CONF_CT_8244US, 246
INA3221_REG_DIE_ID, 245
INA3221_REG_MANUF_ID, 245
INA3221_REG_MASK_ENABLE, 245
INA3221_REG_PWR_VALID_HI_LIM, 245
INA3221_REG_PWR_VALID_LO_LIM, 245
INA3221_REG_SHUNTV_SUM, 245
INA3221_REG_SHUNTV_SUM_LIM, 245
ina3221_reg_t, 245
SHUNT_VOLTAGE_LSB_UV, 246
INA3221::conf_reg_t, 90
 avg_mode, 91
 bus_conv_time, 91
 ch1_en, 91
 ch2_en, 91
 ch3_en, 91
 mode_bus_en, 90
 mode_continious_en, 90
 mode_shunt_en, 90
 reset, 91
 shunt_conv_time, 91
INA3221::masken_reg_t, 128
 conv_ready, 128
 crit_alert_ch1, 129
 crit_alert_ch2, 129
 crit_alert_ch3, 129
 crit_alert_latch_en, 130
 pwr_valid_alert, 129
 reserved, 130
 shunt_sum_alert, 129
 shunt_sum_en_ch1, 130
 shunt_sum_en_ch2, 130
 shunt_sum_en_ch3, 130
 timing_ctrl_alert, 128
 warn_alert_ch1, 129
 warn_alert_ch2, 129
 warn_alert_ch3, 129
 warn_alert_latch_en, 130
ina3221_
 PowerManager, 140
INA3221_ADDR40_GND
 INA3221.h, 245
INA3221_ADDR41_VCC
 INA3221.h, 245
INA3221_ADDR42_SDA
 INA3221.h, 245
INA3221_ADDR43_SCL
 INA3221.h, 245
ina3221_addr_t
 INA3221.h, 244
ina3221_avg_mode_t
 INA3221.h, 246
INA3221_CH1
 INA3221.h, 245
INA3221_CH2
 INA3221.h, 245
INA3221_CH3
 INA3221.h, 245
INA3221_CH_NUM
 INA3221.h, 246
ina3221_ch_t
 INA3221.h, 245
ina3221_conv_time_t
 INA3221.h, 245
INA3221_REG_CH1_BUSV
 INA3221.h, 245
INA3221_REG_CH1_CRIT_ALERT_LIM
 INA3221.h, 245
INA3221_REG_CH1_SHUNTV
 INA3221.h, 245
INA3221_REG_CH1_WARNING_ALERT_LIM
 INA3221.h, 245
INA3221_REG_CH2_SHUNTV
 INA3221.h, 245
INA3221_REG_CH2_WARNING_ALERT_LIM
 INA3221.h, 245
INA3221_REG_CH3_BUSV
 INA3221.h, 245
INA3221_REG_CH3_CRIT_ALERT_LIM
 INA3221.h, 245

INA3221_REG_CH3_SHUNTV
 INA3221.h, 245

INA3221_REG_CH3_WARNING_ALERT_LIM
 INA3221.h, 245

INA3221_REG_CONF
 INA3221.h, 245

INA3221_REG_CONF_AVG_1
 INA3221.h, 246

INA3221_REG_CONF_AVG_1024
 INA3221.h, 246

INA3221_REG_CONF_AVG_128
 INA3221.h, 246

INA3221_REG_CONF_AVG_16
 INA3221.h, 246

INA3221_REG_CONF_AVG_256
 INA3221.h, 246

INA3221_REG_CONF_AVG_4
 INA3221.h, 246

INA3221_REG_CONF_AVG_512
 INA3221.h, 246

INA3221_REG_CONF_AVG_64
 INA3221.h, 246

INA3221_REG_CONF_CT_1100US
 INA3221.h, 246

INA3221_REG_CONF_CT_140US
 INA3221.h, 246

INA3221_REG_CONF_CT_204US
 INA3221.h, 246

INA3221_REG_CONF_CT_2116US
 INA3221.h, 246

INA3221_REG_CONF_CT_332US
 INA3221.h, 246

INA3221_REG_CONF_CT_4156US
 INA3221.h, 246

INA3221_REG_CONF_CT_588US
 INA3221.h, 246

INA3221_REG_CONF_CT_8244US
 INA3221.h, 246

INA3221_REG_DIE_ID
 INA3221.h, 245

INA3221_REG_MANUF_ID
 INA3221.h, 245

INA3221_REG_MASK_ENABLE
 INA3221.h, 245

INA3221_REG_PWR_VALID_HI_LIM
 INA3221.h, 245

INA3221_REG_PWR_VALID_LO_LIM
 INA3221.h, 245

INA3221_REG_SHUNTV_SUM
 INA3221.h, 245

INA3221_REG_SHUNTV_SUM_LIM
 INA3221.h, 245

ina3221_reg_t
 INA3221.h, 245

includes.h, 146

INFO
 utils.h, 295

init

BH1750Wrapper, 73

BME280, 77

BME280Wrapper, 89

EventManager, 106

HMC5883L, 116

HMC5883LWrapper, 119

ISensor, 127

MPU6050Wrapper, 132

init_systems
 main.cpp, 300

initialize
 PowerManager, 137

initialize_radio
 communication.cpp, 179

 communication.h, 183

initialized_
 BH1750Wrapper, 74

 BME280, 79

 BME280Wrapper, 89

 HMC5883LWrapper, 120

 MPU6050Wrapper, 133

 PowerManager, 140

Interface
 protocol.h, 193

interval
 communication.cpp, 181

INVALID_OPERATION
 protocol.h, 193

INVALID_PARAM
 protocol.h, 193

is_charging_solar
 PowerManager, 138

is_charging_usb
 PowerManager, 139

is_initialized
 BH1750Wrapper, 74

 BME280Wrapper, 89

 HMC5883LWrapper, 119

 ISensor, 127

 MPU6050Wrapper, 132

is_open
 FileHandle, 112

ISensor, 126

 ~ISensor, 127

 configure, 127

 get_type, 127

 init, 127

 is_initialized, 127

 read_data, 127

ISensor.h
 ACCEL_X, 276

 ACCEL_Y, 276

 ACCEL_Z, 276

 ENVIRONMENT, 276

 GYRO_X, 276

 GYRO_Y, 276

 GYRO_Z, 276

 HUMIDITY, 276

IMU, 276
LIGHT, 276
LIGHT_LEVEL, 276
MAG_FIELD_X, 276
MAG_FIELD_Y, 276
MAG_FIELD_Z, 276
MAGNETOMETER, 276
PRESSURE, 276
SensorDataTypelIdentifier, 276
SensorType, 275
TEMPERATURE, 276

LAST_SYNC_TIME
 clock_commands.cpp, 156

lastPowerState
 Event Manager, 48

lastPrintTime
 communication.cpp, 181

lastReceiveTime
 communication.cpp, 181

lastSendTime
 communication.cpp, 180

lastSolarState
 Event Manager, 49

lastUSBState
 Event Manager, 49

lib/clock/DS3231.cpp, 147
lib/clock/DS3231.h, 151, 154

lib/comms/commands/clock_commands.cpp, 154, 156
lib/comms/commands/commands.cpp, 158, 159
lib/comms/commands/commands.h, 160, 162
lib/comms/commands/diagnostic_commands.cpp, 162, 163
lib/comms/commands/event_commands.cpp, 164, 165
lib/comms/commands/gps_commands.cpp, 166
lib/comms/commands/power_commands.cpp, 169
lib/comms/commands/storage_commands.cpp, 171, 173
lib/comms/commands/storage_commands_utils.cpp, 175, 177
lib/comms/commands/storage_commands_utils.h, 177, 179
lib/comms/communication.cpp, 179, 181
lib/comms/communication.h, 182, 187
lib/comms/frame.cpp, 188, 189
lib/comms/protocol.h, 191, 198
lib/comms/receive.cpp, 199, 201
lib/comms/send.cpp, 202, 206
lib/comms/utils_converters.cpp, 207, 211
lib/eventman/event_manager.cpp, 212, 213
lib/eventman/event_manager.h, 215, 219
lib/location/gps_collector.cpp, 221, 223
lib/location/gps_collector.h, 224, 225
lib/location/NMEA/NMEA_data.cpp, 226
lib/location/NMEA/NMEA_data.h, 227, 228
lib/pin_config.cpp, 229, 230
lib/pin_config.h, 231, 237
lib/powerman/INA3221/INA3221.cpp, 238
lib/powerman/INA3221/INA3221.h, 243, 247

lib/powerman/PowerManager.cpp, 249
lib/powerman/PowerManager.h, 251, 252
lib/sensors/BH1750/BH1750.cpp, 253
lib/sensors/BH1750/BH1750.h, 254, 256
lib/sensors/BH1750/BH1750_WRAPPER.cpp, 256, 257
lib/sensors/BH1750/BH1750_WRAPPER.h, 258, 259
lib/sensors/BME280/BME280.cpp, 259, 260
lib/sensors/BME280/BME280.h, 263, 264
lib/sensors/BME280/BME280_WRAPPER.cpp, 266
lib/sensors/BME280/BME280_WRAPPER.h, 267
lib/sensors/HMC5883L/HMC5883L.cpp, 268
lib/sensors/HMC5883L/HMC5883L.h, 269, 270
lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp, 270, 271
lib/sensors/HMC5883L/HMC5883L_WRAPPER.h, 272, 273

lib/sensors/ISensor.cpp, 273, 274
lib/sensors/ISensor.h, 274, 276
lib/sensors/MPU6050/MPU6050.cpp, 277
lib/sensors/MPU6050/MPU6050.h, 278
lib/sensors/MPU6050/MPU6050_WRAPPER.cpp, 278
lib/sensors/MPU6050/MPU6050_WRAPPER.h, 278, 279

lib/storage/storage.cpp, 280, 282
lib/storage/storage.h, 282, 285
lib/utils.cpp, 286, 291
lib/utils.h, 292, 298

LIGHT
 ISensor.h, 276

LIGHT_LEVEL
 ISensor.h, 276

LIST_FILES_COMMAND
 storage_commands.cpp, 172

load_from_storage
 EventManager, 107
 EventManagerImpl, 111

LOCK
 Event Manager, 45

log_event
 Event Manager, 46

LOG_FILENAME
 main.cpp, 299

LORA
 protocol.h, 194

lora_address_local
 pin_config.cpp, 230
 pin_config.h, 236

lora_address_remote
 pin_config.cpp, 230
 pin_config.h, 236

lora_cs_pin
 pin_config.cpp, 229
 pin_config.h, 236

LORA_DEFAULT_DIO0_PIN
 pin_config.h, 235

LORA_DEFAULT_RESET_PIN
 pin_config.h, 235

LORA_DEFAULT_SPI

pin_config.h, 235
LORA_DEFAULT_SPI_FREQUENCY
 pin_config.h, 235
LORA_DEFAULT_SS_PIN
 pin_config.h, 235
lora_irq_pin
 pin_config.cpp, 230
 pin_config.h, 236
lora_reset_pin
 pin_config.cpp, 229
 pin_config.h, 236
LOST
 Event Manager, 45
LOW_BATTERY
 Event Manager, 44
MAG_FIELD_X
 ISensor.h, 276
MAG_FIELD_Y
 ISensor.h, 276
MAG_FIELD_Z
 ISensor.h, 276
MAGNETOMETER
 ISensor.h, 276
main
 main.cpp, 301
 test_runner.cpp, 321
main.cpp, 298
 buffer, 302
 bufferIndex, 302
 core1_entry, 299
 init_systems, 300
 LOG_FILENAME, 299
 main, 301
 powerManager, 301
 process_pending_actions, 299
 systemClock, 301
MAIN_I2C_PORT
 pin_config.h, 232
MAIN_I2C_SCL_PIN
 pin_config.h, 233
MAIN_I2C_SDA_PIN
 pin_config.h, 232
MAX_BLOCK_SIZE
 storage_commands.cpp, 172
MAX_RAW_DATA_LENGTH
 gps_collector.cpp, 221
Measurement Functions, 60
 get_current_ma, 61
 get_shunt_voltage, 60
 get_voltage, 62
MILIAMP
 protocol.h, 193
minutes
 ds3231_data_t, 100
Mode
 BH1750, 69
mode_bus_en
 INA3221::conf_reg_t, 90
mode_continious_en
 INA3221::conf_reg_t, 90
mode_shunt_en
 INA3221::conf_reg_t, 90
MONDAY
 DS3231.h, 153
month
 ds3231_data_t, 100
MOUNT_COMMAND
 storage_commands.cpp, 172
MPU6050Wrapper, 131
 configure, 132
 get_type, 132
 init, 132
 initialized_, 133
 is_initialized, 132
 MPU6050Wrapper, 132
 read_data, 132
 sensor_, 133
MSG RECEIVED
 Event Manager, 45
MSG SENT
 Event Manager, 45
msgCount
 communication.cpp, 180
needsPersistence
 EventManager, 108
nextEventId
 EventManager, 108
nmea_data
 gps_collector.cpp, 223
 NMEA_data.cpp, 226
 NMEA_data.h, 228
NMEA_data.cpp
 nmea_data, 226
NMEA_data.h
 nmea_data, 228
NMEAData, 133
 get_gga_tokens, 134
 get_rmc_tokens, 134
 gga_mutex_, 135
 gga_tokens_, 134
 NMEAData, 134
 rmc_mutex_, 135
 rmc_tokens_, 134
 update_gga_tokens, 134
 update_rmc_tokens, 134
NONE
 protocol.h, 193
NOT_ALLOWED
 protocol.h, 193
NUM_CALIB_PARAMS
 BME280, 84
on_receive
 communication.h, 183
 receive.cpp, 200
ONE_TIME_HIGH_RES_MODE

BH1750, 70
ONE_TIME_HIGH_RES_MODE_2
 BH1750, 70
ONE_TIME_LOW_RES_MODE
 BH1750, 70
operation_type_to_string
 protocol.h, 194
 utils_converters.cpp, 209
OperationType
 protocol.h, 192
operationType
 Frame, 114
outgoing
 communication.cpp, 180
OVERCHARGE
 Event Manager, 44
PA_OUTPUT_PA_BOOST_PIN
 pin_config.h, 236
PA_OUTPUT_RFO_PIN
 pin_config.h, 236
PARAM_UNNECESSARY
 protocol.h, 193
PASS_THROUGH_END
 Event Manager, 45
PASS_THROUGH_START
 Event Manager, 45
pin_config.cpp
 lora_address_local, 230
 lora_address_remote, 230
 lora_cs_pin, 229
 lora_irq_pin, 230
 lora_reset_pin, 229
pin_config.h
 BUFFER_SIZE, 233
 DEBUG_UART_BAUD_RATE, 232
 DEBUG_UART_PORT, 232
 DEBUG_UART_RX_PIN, 232
 DEBUG_UART_TX_PIN, 232
 GPS_POWER_ENABLE_PIN, 233
 GPS_UART_BAUD_RATE, 233
 GPS_UART_PORT, 233
 GPS_UART_RX_PIN, 233
 GPS_UART_TX_PIN, 233
 lora_address_local, 236
 lora_address_remote, 236
 lora_cs_pin, 236
 LORA_DEFAULT_DIO0_PIN, 235
 LORA_DEFAULT_RESET_PIN, 235
 LORA_DEFAULT_SPI, 235
 LORA_DEFAULT_SPI_FREQUENCY, 235
 LORA_DEFAULT_SS_PIN, 235
 lora_irq_pin, 236
 lora_reset_pin, 236
 MAIN_I2C_PORT, 232
 MAIN_I2C_SCL_PIN, 233
 MAIN_I2C_SDA_PIN, 232
 PA_OUTPUT_PA_BOOST_PIN, 236
 PA_OUTPUT_RFO_PIN, 236
READ_BIT, 235
SD_CARD_DETECT_PIN, 234
SD_CS_PIN, 234
SD_MISO_PIN, 234
SD_MOSI_PIN, 234
SD_SCK_PIN, 234
SD_SPI_PORT, 233
SPI_PORT, 235
SX1278_CS, 234
SX1278_MISO, 234
SX1278_MOSI, 235
SX1278_SCK, 234
POWER
 Event Manager, 44
Power Commands, 27
 handle_get_current_charge_solar, 30
 handle_get_current_charge_total, 31
 handle_get_current_charge_usb, 30
 handle_get_current_draw, 32
 handle_get_power_manager_ids, 28
 handle_get_voltage_5v, 29
 handle_get_voltage_battery, 28
POWER_FALLING
 Event Manager, 44
POWER_NORMAL
 Event Manager, 44
POWER_OFF
 Event Manager, 45
POWER_ON
 BH1750, 70
 Event Manager, 45
PowerEvent
 Event Manager, 44
powerman_mutex_
 PowerManager, 141
PowerManager, 135
 charging_solar_active_, 141
 charging_usb_active_, 141
 check_power_alerts, 139
 configure, 138
 FALL_RATE_THRESHOLD, 140
 FALLING_TREND_REQUIRED, 140
 get_current_charge_solar, 137
 get_current_charge_total, 137
 get_current_charge_usb, 137
 get_current_draw, 137
 get_voltage_5v, 138
 get_voltage_battery, 138
 ina3221_, 140
 initialize, 137
 initialized_, 140
 is_charging_solar, 138
 is_charging_usb, 139
 powerman_mutex_, 141
 PowerManager, 136
 read_device_ids, 137
 SOLAR_CURRENT_THRESHOLD, 140
 USB_CURRENT_THRESHOLD, 140

VOLTAGE_LOW_THRESHOLD, 140
 VOLTAGE_OVERCHARGE_THRESHOLD, 140
powerManager
 main.cpp, 301
PRESSURE
 ISensor.h, 276
process_pending_actions
 main.cpp, 299
protocol.h
 BOOL, 193
 CommandAccessLevel, 192
 DATETIME, 193
 DELIMITER, 198
 ERR, 192
 exception_type_to_string, 194
 ExceptionType, 193
 FRAME_BEGIN, 197
 FRAME_END, 197
 GET, 192
 hex_string_to_bytes, 195
 Interface, 193
 INVALID_OPERATION, 193
 INVALID_PARAM, 193
 LORA, 194
 MILIAMP, 193
 NONE, 193
 NOT_ALLOWED, 193
 operation_type_to_string, 194
 OperationType, 192
 PARAM_UNNECESSARY, 193
 READ_ONLY, 193
 READ_WRITE, 193
 RES, 192
 SECOND, 193
 SEQ, 192
 SET, 192
 string_to_operation_type, 195
 TEXT, 193
 UART, 194
 UNDEFINED, 193
 VAL, 192
 value_unit_type_to_string, 196
 ValueUnit, 193
 VOLT, 193
 WRITE_ONLY, 193
pwr_valid_alert
 INA3221::masken_reg_t, 129
RADIO_ERROR
 Event Manager, 45
RADIO_INIT
 Event Manager, 45
read
 HMC5883L, 116
READ_BIT
 pin_config.h, 235
read_data
 BH1750Wrapper, 73
 BME280Wrapper, 89
 HMC5883LWrapper, 119
 ISensor, 127
 MPU6050Wrapper, 132
read_device_ids
 PowerManager, 137
READ_ONLY
 protocol.h, 193
read_raw_all
 BME280, 77
read_register
 Configuration Functions, 54
 HMC5883L, 117
read_temperature
 DS3231, 94
READ_WRITE
 protocol.h, 193
receive.cpp
 handle_uart_input, 200
 on_receive, 200
receive_ack
 storage_commands_utils.cpp, 176
 storage_commands_utils.h, 178
REG_CONFIG
 BME280, 79
REG_CTRL_HUM
 BME280, 79
REG_CTRL_MEAS
 BME280, 79
REG_DIG_H1
 BME280, 83
REG_DIG_H2
 BME280, 83
REG_DIG_H3
 BME280, 83
REG_DIG_H4
 BME280, 83
REG_DIG_H5
 BME280, 84
REG_DIG_H6
 BME280, 84
REG_DIG_P1_LSB
 BME280, 81
REG_DIG_P1_MSB
 BME280, 81
REG_DIG_P2_LSB
 BME280, 81
REG_DIG_P2_MSB
 BME280, 81
REG_DIG_P3_LSB
 BME280, 81
REG_DIG_P3_MSB
 BME280, 81
REG_DIG_P4_LSB
 BME280, 82
REG_DIG_P4_MSB
 BME280, 82
REG_DIG_P5_LSB
 BME280, 82

REG_DIG_P5_MSB
 BME280, 82
REG_DIG_P6_LSB
 BME280, 82
REG_DIG_P6_MSB
 BME280, 82
REG_DIG_P7_LSB
 BME280, 82
REG_DIG_P7_MSB
 BME280, 82
REG_DIG_P8_LSB
 BME280, 83
REG_DIG_P8_MSB
 BME280, 83
REG_DIG_P9_LSB
 BME280, 83
REG_DIG_P9_MSB
 BME280, 83
REG_DIG_T1_LSB
 BME280, 80
REG_DIG_T1_MSB
 BME280, 80
REG_DIG_T2_LSB
 BME280, 80
REG_DIG_T2_MSB
 BME280, 80
REG_DIG_T3_LSB
 BME280, 81
REG_DIG_T3_MSB
 BME280, 81
REG_HUMIDITY_MSB
 BME280, 80
REG_PRESSURE_MSB
 BME280, 80
REG_RESET
 BME280, 80
REG_TEMPERATURE_MSB
 BME280, 80
RES
 protocol.h, 192
reserved
 INA3221::masken_reg_t, 130
RESET
 BH1750, 70
reset
 BME280, 77
 Configuration Functions, 52
 INA3221::conf_reg_t, 91
rmc_mutex_
 NMEAData, 135
rmc_tokens_
 NMEAData, 134

SATURDAY
 DS3231.h, 153
save_to_storage
 EventManager, 107
 EventManagerImpl, 111
SD_CARD_DETECT_PIN
 pin_config.h, 234
sd_card_mounted
 storage.cpp, 282
 storage.h, 285
SD_CS_PIN
 pin_config.h, 234
SD_MISO_PIN
 pin_config.h, 234
SD_MOSI_PIN
 pin_config.h, 234
SD_SCK_PIN
 pin_config.h, 234
SD_SPI_PORT
 pin_config.h, 233
SECOND
 protocol.h, 193
seconds
 ds3231_data_t, 100
send.cpp
 send_frame, 205
 send_frame_lora, 203
 send_frame_uart, 204
 send_message, 203
 split_and_send_message, 205
send_data_block
 storage_commands_utils.cpp, 176
 storage_commands_utils.h, 178
send_frame
 communication.h, 185
 send.cpp, 205
send_frame_lora
 communication.h, 186
 send.cpp, 203
send_frame_uart
 communication.h, 185
 send.cpp, 204
send_message
 communication.h, 185
 send.cpp, 203
sensor_
 BH1750Wrapper, 74
 BME280Wrapper, 89
 HMC5883LWrapper, 120
 MPU6050Wrapper, 133
sensor_configure
 SensorWrapper, 143
sensor_init
 SensorWrapper, 143
sensor_read_data
 SensorWrapper, 144
SensorDataTypeIdentifier
 ISensor.h, 276
sensors
 SensorWrapper, 144
SensorType
 ISensor.h, 275
SensorWrapper
 141
 get_instance, 143

sensor_configure, 143
 sensor_init, 143
 sensor_read_data, 144
 sensors, 144
 SensorWrapper, 142
SEQ
 protocol.h, 192
SET
 protocol.h, 192
set_alert_latch
 Alert Functions, 67
set_averaging_mode
 Configuration Functions, 58
set_bus_conversion_time
 Configuration Functions, 59
set_bus_measurement_disable
 Configuration Functions, 58
set_bus_measurement_enable
 Configuration Functions, 57
set_crit_alert_limit
 Alert Functions, 63
set_mode_continuous
 Configuration Functions, 55
set_mode_power_down
 Configuration Functions, 55
set_mode_triggered
 Configuration Functions, 56
set_power_valid_limit
 Alert Functions, 64
set_shunt_conversion_time
 Configuration Functions, 59
set_shunt_measurement_disable
 Configuration Functions, 57
set_shunt_measurement_enable
 Configuration Functions, 56
set_time
 DS3231, 93
set_unix_time
 DS3231, 94
set_warn_alert_limit
 Alert Functions, 63
shunt_conv_time
 INA3221::conf_reg_t, 91
shunt_sum_alert
 INA3221::masken_reg_t, 129
shunt_sum_en_ch1
 INA3221::masken_reg_t, 130
shunt_sum_en_ch2
 INA3221::masken_reg_t, 130
shunt_sum_en_ch3
 INA3221::masken_reg_t, 130
SHUNT_VOLTAGE_LSB_UV
 INA3221.h, 246
SHUTDOWN
 Event Manager, 44
SILENT
 utils.h, 295
SOLAR_ACTIVE
 Event Manager, 44
SOLAR_CURRENT_THRESHOLD
 PowerManager, 140
SOLAR_INACTIVE
 Event Manager, 44
SPI_PORT
 pin_config.h, 235
split_and_send_message
 communication.h, 187
 send.cpp, 205
splitString
 gps_collector.cpp, 222
START_COMMAND
 storage_commands.cpp, 172
Storage Commands, 33
 handle_file_download, 33
 handle_list_files, 34
 handle_mount, 35
storage.cpp
 fs_init, 281
 sd_card_mounted, 282
storage.h
 fs_close_file, 285
 fs_file_exists, 285
 fs_init, 283
 fs_open_file, 284
 fs_read_file, 284
 fs_write_file, 284
 sd_card_mounted, 285
storage_commands.cpp
 DATA_COMMAND, 172
 END_COMMAND, 172
 LIST_FILES_COMMAND, 172
 MAX_BLOCK_SIZE, 172
 MOUNT_COMMAND, 172
 START_COMMAND, 172
 STORAGE_GROUP, 172
storage_commands_utils.cpp
 calculate_checksum, 176
 receive_ack, 176
 send_data_block, 176
storage_commands_utils.h
 calculate_checksum, 178
 receive_ack, 178
 send_data_block, 178
STORAGE_GROUP
 storage_commands.cpp, 172
string_to_operation_type
 protocol.h, 195
 utils_converters.cpp, 210
SUNDAY
 DS3231.h, 153
SX1278_CS
 pin_config.h, 234
SX1278_MISO
 pin_config.h, 234
SX1278_MOSI
 pin_config.h, 235

SX1278_SCK
pin_config.h, 234

SYSTEM
Event Manager, 44

systemClock
Clock Management Commands, 14
Event Manager, 49
main.cpp, 301

SystemEvent
Event Manager, 44

t_fine
BME280, 79

TEMPERATURE
ISensor.h, 276

test/comms/test_converters.cpp, 304, 307
test/comms/test_frame_build.cpp, 307, 309
test/comms/test_frame_coding.cpp, 310, 312
test/comms/test_frame_common.h, 313, 314
test/test_runner.cpp, 314, 321
test_converters.cpp
test_exception_type_conversion, 305
test_hex_string_conversion, 306
test_operation_type_conversion, 304
test_value_unit_type_conversion, 305

test_exception_type_conversion
test_converters.cpp, 305
test_runner.cpp, 319

test_frame_build.cpp
test_frame_build_error, 308
test_frame_build_info, 309
test_frame_build_success, 308

test_frame_build_error
test_frame_build.cpp, 308
test_runner.cpp, 317

test_frame_build_info
test_frame_build.cpp, 309
test_runner.cpp, 317

test_frame_build_success
test_frame_build.cpp, 308
test_runner.cpp, 316

test_frame_coding.cpp
test_frame_decode_basic, 311
test_frame_decode_invalid_header, 311
test_frame_encode_basic, 310

test_frame_common.h
create_test_frame, 313

test_frame_decode_basic
test_frame_coding.cpp, 311
test_runner.cpp, 315

test_frame_decode_invalid_header
test_frame_coding.cpp, 311
test_runner.cpp, 316

test_frame_encode_basic
test_frame_coding.cpp, 310
test_runner.cpp, 315

test_hex_string_conversion
test_converters.cpp, 306
test_runner.cpp, 320

test_operation_type_conversion
test_converters.cpp, 304
test_runner.cpp, 318

test_runner.cpp
main, 321
test_exception_type_conversion, 319
test_frame_build_error, 317
test_frame_build_info, 317
test_frame_build_success, 316
test_frame_decode_basic, 315
test_frame_decode_invalid_header, 316
test_frame_encode_basic, 315
test_hex_string_conversion, 320
test_operation_type_conversion, 318
test_value_unit_type_conversion, 319

test_value_unit_type_conversion
test_converters.cpp, 305
test_runner.cpp, 319

TEXT
protocol.h, 193

THURSDAY
DS3231.h, 153

TIME
clock_commands.cpp, 155

timestamp
event_manager.h, 218
EventLog, 103

TIMEZONE_OFFSET
clock_commands.cpp, 155

timing_ctrl_alert
INA3221::masken_reg_t, 128

to_string
event_manager.h, 217
EventLog, 103

TUESDAY
DS3231.h, 153

UART
protocol.h, 194

UART_ERROR
Event Manager, 45

uart_mutex
utils.cpp, 291

uart_print
utils.cpp, 289
utils.h, 295

UNCONFIGURED_POWER_DOWN
BH1750, 70

UNDEFINED
protocol.h, 193

unit
Frame, 114

update_gga_tokens
NMEAData, 134

update_rmc_tokens
NMEAData, 134

USB_CONNECTED
Event Manager, 44

USB_CURRENT_THRESHOLD

PowerManager, 140
USB_DISCONNECTED
 Event Manager, 44
utils.cpp
 crc16, 290
 g_uart_verbosity, 291
 get_level_color, 287
 get_level_prefix, 288
 uart_mutex, 291
 uart_print, 289
utils.h
 ANSI_BLUE, 294
 ANSI_CYAN, 294
 ANSI_GREEN, 294
 ANSI_RED, 294
 ANSI_RESET, 294
 ANSI_YELLOW, 294
 crc16, 296
 DEBUG, 295
 ERROR, 295
 EVENT, 295
 g_uart_verbosity, 297
 INFO, 295
 SILENT, 295
 uart_print, 295
 VerbosityLevel, 294
 WARNING, 295
utils_converters.cpp
 exception_type_to_string, 207
 hex_string_to_bytes, 210
 operation_type_to_string, 209
 string_to_operation_type, 210
 value_unit_type_to_string, 208

VAL
 protocol.h, 192
value
 Frame, 114
value_unit_type_to_string
 protocol.h, 196
 utils_converters.cpp, 208
ValueUnit
 protocol.h, 193
VerbosityLevel
 utils.h, 294
VOLT
 protocol.h, 193
VOLTAGE_LOW_THRESHOLD
 Event Manager, 48
 PowerManager, 140
VOLTAGE_OVERCHARGE_THRESHOLD
 Event Manager, 48
 PowerManager, 140

warn_alert_ch1
 INA3221::masken_reg_t, 129
warn_alert_ch2
 INA3221::masken_reg_t, 129
warn_alert_ch3
 INA3221::masken_reg_t, 129
warn_alert_latch_en
 INA3221::masken_reg_t, 130
WARNING
 utils.h, 295
WATCHDOG_RESET
 Event Manager, 44
WEDNESDAY
 DS3231.h, 153
write8
 BH1750, 71
WRITE_ONLY
 protocol.h, 193
write_register
 HMC5883L, 116
writeIndex
 EventManager, 108
year
 ds3231_data_t, 100