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 . . . 15

6.1.3.1 systemClock . . . 15

6.2 Command System . . . 15

6.2.1 Detailed Description . . . 15

6.2.2 Typedef Documentation . . . 15

6.2.2.1 CommandHandler . . . 15

6.2.2.2 CommandMap . . . 16

6.2.3 Function Documentation . . . 16

6.2.3.1 execute_command() . . . 16

6.2.4 Variable Documentation . . . 17

6.2.4.1 command_handlers . . . 17

6.3 Diagnostic Commands . . . 17

6.3.1 Detailed Description . . . 18

6.3.2 Function Documentation . . . 18

6.3.2.1 handle_get_commands_list() . . . 18

6.3.2.2 handle_get_build_version() . . . 18

6.3.2.3 handle_verbosity() . . . 19

6.3.2.4 handle_enter_bootloader_mode() . . . 21

6.3.3 Variable Documentation . . . 21

6.3.3.1 g_pending_bootloader_reset . . . 21

6.4 Event Commands . . . 21

6.4.1 Detailed Description . . . 22

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

6.10.2.4 frame_build() . . . 45
6.11 Event Manager . . . 46
6.11.1 Detailed Description . . . 48
6.11.2 Enumeration Type Documentation . . . 48
6.11.2.1 EventGroup . . . 48
6.11.2.2 SystemEvent . . . 48
6.11.2.3 PowerEvent . . . 49
6.11.2.4 CommsEvent . . . 49
6.11.2.5 GPSEvent . . . 49
6.11.2.6 ClockEvent . . . 50
6.11.3 Function Documentation . . . 50
6.11.3.1 __attribute__() . . . 50
6.11.3.2 log_event() . . . 50
6.11.3.3 get_event() . . . 51
6.11.4 Variable Documentation . . . 52
6.11.4.1 eventLogId . . . 52
6.11.4.2 systemClock . . . 52
6.11.4.3 eventManager [1/2] . . . 52
6.11.4.4 __attribute__ . . . 52
6.11.4.5 eventManager [2/2] . . . 52
6.12 INA3221 Power Monitor . . . 53
6.12.1 Detailed Description . . . 53
6.12.2 Configuration Functions . . . 53
6.12.2.1 Detailed Description . . . 54
6.12.2.2 Function Documentation . . . 54
6.12.3 Measurement Functions . . . 63
6.12.3.1 Detailed Description . . . 63
6.12.3.2 Function Documentation . . . 63
6.12.4 Alert Functions . . . 65
6.12.4.1 Detailed Description . . . 66
6.12.4.2 Function Documentation . . . 66
6.13 Telemetry Manager . . . 71
6.13.1 Detailed Description . . . 72
6.13.2 Function Documentation . . . 72
6.13.2.1 telemetry_init() . . . 72
6.13.2.2 collect_telemetry() . . . 73
6.13.2.3 flush_telemetry() . . . 74
6.13.2.4 flush_sensor_data() . . . 74
6.13.2.5 is_telemetry_collection_time() . . . 75
6.13.2.6 is_telemetry_flush_time() . . . 76
6.13.2.7 get_telemetry_sample_interval() . . . 77
6.13.2.8 set_telemetry_sample_interval() . . . 77

6.13.2.9 get_telemetry_flush_threshold() . . . 77
6.13.2.10 set_telemetry_flush_threshold() . . . 77
6.13.2.11 get_last_telemetry_record_csv() . . . 78
6.13.2.12 get_last_sensor_record_csv() . . . 78
6.13.3 Variable Documentation . . . 79
6.13.3.1 TELEMETRY_BUFFER_SIZE [1/2] . . . 79
6.13.3.2 telemetry_buffer . . . 79
6.13.3.3 telemetry_buffer_count . . . 79
6.13.3.4 telemetry_buffer_write_index . . . 80
6.13.3.5 sensor_data_buffer . . . 80
6.13.3.6 sensor_data_buffer_count . . . 80
6.13.3.7 sensor_data_buffer_write_index . . . 80
6.13.3.8 telemetry_mutex . . . 80
6.13.3.9 TELEMETRY_BUFFER_SIZE [2/2] . . . 80

7 Class Documentation **81**
7.1 BH1750 Class Reference . . . 81
7.1.1 Detailed Description . . . 81
7.1.2 Member Enumeration Documentation . . . 81
7.1.2.1 Mode . . . 81
7.1.3 Constructor & Destructor Documentation . . . 82
7.1.3.1 BH1750() . . . 82
7.1.4 Member Function Documentation . . . 82
7.1.4.1 begin() . . . 82
7.1.4.2 configure() . . . 83
7.1.4.3 get_light_level() . . . 83
7.1.4.4 write8() . . . 83
7.1.5 Member Data Documentation . . . 84
7.1.5.1 _i2c_addr . . . 84
7.2 BH1750Wrapper Class Reference . . . 84
7.2.1 Detailed Description . . . 85
7.2.2 Constructor & Destructor Documentation . . . 85
7.2.2.1 BH1750Wrapper() . . . 85
7.2.3 Member Function Documentation . . . 85
7.2.3.1 get_i2c_addr() . . . 85
7.2.3.2 init() . . . 85
7.2.3.3 read_data() . . . 86
7.2.3.4 is_initialized() . . . 86
7.2.3.5 get_type() . . . 86
7.2.3.6 configure() . . . 86
7.2.3.7 get_address() . . . 86
7.2.4 Member Data Documentation . . . 86

7.2.4.1 sensor_ . . . 86
7.2.4.2 initialized_ . . . 87
7.3 BME280 Class Reference . . . 87
7.3.1 Detailed Description . . . 88
7.3.2 Constructor & Destructor Documentation . . . 89
7.3.2.1 BME280() . . . 89
7.3.3 Member Function Documentation . . . 89
7.3.3.1 init() . . . 89
7.3.3.2 reset() . . . 89
7.3.3.3 read_raw_all() . . . 89
7.3.3.4 convert_temperature() . . . 90
7.3.3.5 convert_pressure() . . . 90
7.3.3.6 convert_humidity() . . . 90
7.3.3.7 configure_sensor() . . . 90
7.3.3.8 get_calibration_parameters() . . . 90
7.3.4 Member Data Documentation . . . 91
7.3.4.1 ADDR_SDO_LOW . . . 91
7.3.4.2 ADDR_SDO_HIGH . . . 91
7.3.4.3 i2c_port . . . 91
7.3.4.4 device_addr . . . 91
7.3.4.5 calib_params . . . 91
7.3.4.6 initialized_ . . . 91
7.3.4.7 t_fine . . . 91
7.3.4.8 REG_CONFIG . . . 92
7.3.4.9 REG_CTRL_MEAS . . . 92
7.3.4.10 REG_CTRL_HUM . . . 92
7.3.4.11 REG_RESET . . . 92
7.3.4.12 REG_PRESSURE_MSB . . . 92
7.3.4.13 REG_TEMPERATURE_MSB . . . 92
7.3.4.14 REG_HUMIDITY_MSB . . . 92
7.3.4.15 REG_DIG_T1_LSB . . . 92
7.3.4.16 REG_DIG_T1_MSB . . . 93
7.3.4.17 REG_DIG_T2_LSB . . . 93
7.3.4.18 REG_DIG_T2_MSB . . . 93
7.3.4.19 REG_DIG_T3_LSB . . . 93
7.3.4.20 REG_DIG_T3_MSB . . . 93
7.3.4.21 REG_DIG_P1_LSB . . . 93
7.3.4.22 REG_DIG_P1_MSB . . . 93
7.3.4.23 REG_DIG_P2_LSB . . . 93
7.3.4.24 REG_DIG_P2_MSB . . . 94
7.3.4.25 REG_DIG_P3_LSB . . . 94
7.3.4.26 REG_DIG_P3_MSB . . . 94

7.3.4.27 REG_DIG_P4_LSB . . . 94
7.3.4.28 REG_DIG_P4_MSB . . . 94
7.3.4.29 REG_DIG_P5_LSB . . . 94
7.3.4.30 REG_DIG_P5_MSB . . . 94
7.3.4.31 REG_DIG_P6_LSB . . . 94
7.3.4.32 REG_DIG_P6_MSB . . . 95
7.3.4.33 REG_DIG_P7_LSB . . . 95
7.3.4.34 REG_DIG_P7_MSB . . . 95
7.3.4.35 REG_DIG_P8_LSB . . . 95
7.3.4.36 REG_DIG_P8_MSB . . . 95
7.3.4.37 REG_DIG_P9_LSB . . . 95
7.3.4.38 REG_DIG_P9_MSB . . . 95
7.3.4.39 REG_DIG_H1 . . . 95
7.3.4.40 REG_DIG_H2 . . . 96
7.3.4.41 REG_DIG_H3 . . . 96
7.3.4.42 REG_DIG_H4 . . . 96
7.3.4.43 REG_DIG_H5 . . . 96
7.3.4.44 REG_DIG_H6 . . . 96
7.3.4.45 NUM_CALIB_PARAMS . . . 96
7.4 BME280CalibParam Struct Reference . . . 96
7.4.1 Detailed Description . . . 97
7.4.2 Member Data Documentation . . . 97
7.4.2.1 dig_t1 . . . 97
7.4.2.2 dig_t2 . . . 97
7.4.2.3 dig_t3 . . . 97
7.4.2.4 dig_p1 . . . 98
7.4.2.5 dig_p2 . . . 98
7.4.2.6 dig_p3 . . . 98
7.4.2.7 dig_p4 . . . 98
7.4.2.8 dig_p5 . . . 98
7.4.2.9 dig_p6 . . . 98
7.4.2.10 dig_p7 . . . 98
7.4.2.11 dig_p8 . . . 98
7.4.2.12 dig_p9 . . . 99
7.4.2.13 dig_h1 . . . 99
7.4.2.14 dig_h2 . . . 99
7.4.2.15 dig_h3 . . . 99
7.4.2.16 dig_h4 . . . 99
7.4.2.17 dig_h5 . . . 99
7.4.2.18 dig_h6 . . . 99
7.5 BME280Wrapper Class Reference . . . 100
7.5.1 Detailed Description . . . 101

7.5.2 Constructor & Destructor Documentation . . . 101
7.5.2.1 BME280Wrapper() . . . 101
7.5.3 Member Function Documentation . . . 101
7.5.3.1 init() . . . 101
7.5.3.2 read_data() . . . 101
7.5.3.3 is_initialized() . . . 101
7.5.3.4 get_type() . . . 102
7.5.3.5 configure() . . . 102
7.5.3.6 get_address() . . . 102
7.5.4 Member Data Documentation . . . 102
7.5.4.1 sensor_ . . . 102
7.5.4.2 initialized_ . . . 102
7.6 INA3221::conf_reg_t Struct Reference . . . 102
7.6.1 Detailed Description . . . 103
7.6.2 Member Data Documentation . . . 103
7.6.2.1 mode_shunt_en . . . 103
7.6.2.2 mode_bus_en . . . 103
7.6.2.3 mode_continious_en . . . 103
7.6.2.4 shunt_conv_time . . . 103
7.6.2.5 bus_conv_time . . . 104
7.6.2.6 avg_mode . . . 104
7.6.2.7 ch3_en . . . 104
7.6.2.8 ch2_en . . . 104
7.6.2.9 ch1_en . . . 104
7.6.2.10 reset . . . 104
7.7 DS3231 Class Reference . . . 104
7.7.1 Detailed Description . . . 106
7.7.2 Constructor & Destructor Documentation . . . 106
7.7.2.1 DS3231() . . . 106
7.7.3 Member Function Documentation . . . 106
7.7.3.1 set_time() . . . 106
7.7.3.2 get_time() . . . 107
7.7.3.3 read_temperature() . . . 109
7.7.3.4 set_unix_time() . . . 110
7.7.3.5 get_unix_time() . . . 111
7.7.3.6 clock_enable() . . . 112
7.7.3.7 get_timezone_offset() . . . 113
7.7.3.8 set_timezone_offset() . . . 113
7.7.3.9 get_clock_sync_interval() . . . 114
7.7.3.10 set_clock_sync_interval() . . . 114
7.7.3.11 get_last_sync_time() . . . 115
7.7.3.12 update_last_sync_time() . . . 116

7.7.3.13 get_local_time() . . . 116
7.7.3.14 is_sync_needed() . . . 117
7.7.3.15 sync_clock_with_gps() . . . 117
7.7.3.16 i2c_read_reg() . . . 118
7.7.3.17 i2c_write_reg() . . . 119
7.7.3.18 bin_to_bcd() . . . 121
7.7.3.19 bcd_to_bin() . . . 121
7.7.4 Member Data Documentation . . . 122
7.7.4.1 i2c . . . 122
7.7.4.2 ds3231_addr . . . 122
7.7.4.3 clock_mutex_ . . . 123
7.7.4.4 timezone_offset_minutes_ . . . 123
7.7.4.5 sync_interval_minutes_ . . . 123
7.7.4.6 last_sync_time_ . . . 123
7.8 ds3231_data_t Struct Reference . . . 123
7.8.1 Detailed Description . . . 124
7.8.2 Member Data Documentation . . . 124
7.8.2.1 seconds . . . 124
7.8.2.2 minutes . . . 124
7.8.2.3 hours . . . 124
7.8.2.4 day . . . 125
7.8.2.5 date . . . 125
7.8.2.6 month . . . 125
7.8.2.7 year . . . 125
7.8.2.8 century . . . 125
7.9 EventEmitter Class Reference . . . 125
7.9.1 Detailed Description . . . 126
7.9.2 Member Function Documentation . . . 126
7.9.2.1 emit() . . . 126
7.10 EventLog Class Reference . . . 127
7.10.1 Detailed Description . . . 128
7.10.2 Member Function Documentation . . . 128
7.10.2.1 to_string() . . . 128
7.10.3 Member Data Documentation . . . 128
7.10.3.1 id . . . 128
7.10.3.2 timestamp . . . 128
7.10.3.3 group . . . 129
7.10.3.4 event . . . 129
7.11 EventManager Class Reference . . . 129
7.11.1 Detailed Description . . . 130
7.11.2 Constructor & Destructor Documentation . . . 130
7.11.2.1 EventManager() . . . 130

7.11.2.2 ~EventManager() . . . 131
7.11.3 Member Function Documentation . . . 131
7.11.3.1 init() . . . 131
7.11.3.2 get_event_count() . . . 131
7.11.3.3 save_to_storage() . . . 132
7.11.3.4 load_from_storage() . . . 132
7.11.4 Member Data Documentation . . . 132
7.11.4.1 events . . . 132
7.11.4.2 eventCount . . . 133
7.11.4.3 writeIndex . . . 133
7.11.4.4 eventMutex . . . 133
7.11.4.5 nextEventId . . . 133
7.11.4.6 eventsSinceFlush . . . 133
7.12 EventManagerImpl Class Reference . . . 134
7.12.1 Detailed Description . . . 135
7.12.2 Constructor & Destructor Documentation . . . 136
7.12.2.1 EventManagerImpl() . . . 136
7.12.3 Member Function Documentation . . . 136
7.12.3.1 save_to_storage() . . . 136
7.12.3.2 load_from_storage() . . . 137
7.13 FileHandle Struct Reference . . . 137
7.13.1 Detailed Description . . . 137
7.13.2 Member Data Documentation . . . 137
7.13.2.1 fd . . . 137
7.13.2.2 is_open . . . 137
7.14 Frame Struct Reference . . . 138
7.14.1 Detailed Description . . . 138
7.14.2 Member Data Documentation . . . 139
7.14.2.1 header . . . 139
7.14.2.2 direction . . . 139
7.14.2.3 operationType . . . 139
7.14.2.4 group . . . 139
7.14.2.5 command . . . 140
7.14.2.6 value . . . 140
7.14.2.7 unit . . . 140
7.14.2.8 footer . . . 140
7.15 HMC5883L Class Reference . . . 140
7.15.1 Detailed Description . . . 141
7.15.2 Constructor & Destructor Documentation . . . 141
7.15.2.1 HMC5883L() . . . 141
7.15.3 Member Function Documentation . . . 141
7.15.3.1 init() . . . 141

7.15.3.2 read() . . . 142
7.15.3.3 write_register() . . . 142
7.15.3.4 read_register() . . . 142
7.15.4 Member Data Documentation . . . 143
7.15.4.1 i2c . . . 143
7.15.4.2 address . . . 143
7.16 HMC5883LWrapper Class Reference . . . 143
7.16.1 Detailed Description . . . 144
7.16.2 Constructor & Destructor Documentation . . . 144
7.16.2.1 HMC5883LWrapper() . . . 144
7.16.3 Member Function Documentation . . . 145
7.16.3.1 init() . . . 145
7.16.3.2 read_data() . . . 145
7.16.3.3 is_initialized() . . . 145
7.16.3.4 get_type() . . . 145
7.16.3.5 configure() . . . 145
7.16.3.6 get_address() . . . 145
7.16.4 Member Data Documentation . . . 146
7.16.4.1 sensor_ . . . 146
7.16.4.2 initialized_ . . . 146
7.17 INA3221 Class Reference . . . 146
7.17.1 Detailed Description . . . 148
7.17.2 Member Function Documentation . . . 148
7.17.2.1 _read() . . . 148
7.17.2.2 _write() . . . 150
7.17.2.3 get_current() . . . 151
7.17.3 Member Data Documentation . . . 152
7.17.3.1 _i2c . . . 152
7.17.3.2 _i2c_addr . . . 152
7.17.3.3 _shuntRes . . . 152
7.17.3.4 _filterRes . . . 152
7.17.3.5 _masken_reg . . . 152
7.18 ISensor Class Reference . . . 152
7.18.1 Detailed Description . . . 153
7.18.2 Constructor & Destructor Documentation . . . 153
7.18.2.1 ~ISensor() . . . 153
7.18.3 Member Function Documentation . . . 153
7.18.3.1 init() . . . 153
7.18.3.2 read_data() . . . 153
7.18.3.3 is_initialized() . . . 153
7.18.3.4 get_type() . . . 153
7.18.3.5 configure() . . . 154

7.18.3.6 get_address() . . . 154
7.19 INA3221::masken_reg_t Struct Reference . . . 154
7.19.1 Detailed Description . . . 154
7.19.2 Member Data Documentation . . . 155
7.19.2.1 conv_ready . . . 155
7.19.2.2 timing_ctrl_alert . . . 155
7.19.2.3 pwr_valid_alert . . . 155
7.19.2.4 warn_alert_ch3 . . . 155
7.19.2.5 warn_alert_ch2 . . . 155
7.19.2.6 warn_alert_ch1 . . . 155
7.19.2.7 shunt_sum_alert . . . 155
7.19.2.8 crit_alert_ch3 . . . 156
7.19.2.9 crit_alert_ch2 . . . 156
7.19.2.10 crit_alert_ch1 . . . 156
7.19.2.11 crit_alert_latch_en . . . 156
7.19.2.12 warn_alert_latch_en . . . 156
7.19.2.13 shunt_sum_en_ch3 . . . 156
7.19.2.14 shunt_sum_en_ch2 . . . 156
7.19.2.15 shunt_sum_en_ch1 . . . 156
7.19.2.16 reserved . . . 157
7.20 MPU6050Wrapper Class Reference . . . 157
7.20.1 Detailed Description . . . 158
7.20.2 Constructor & Destructor Documentation . . . 158
7.20.2.1 MPU6050Wrapper() . . . 158
7.20.3 Member Function Documentation . . . 158
7.20.3.1 init() . . . 158
7.20.3.2 read_data() . . . 158
7.20.3.3 is_initialized() . . . 158
7.20.3.4 get_type() . . . 158
7.20.3.5 configure() . . . 159
7.20.4 Member Data Documentation . . . 159
7.20.4.1 sensor_ . . . 159
7.20.4.2 initialized_ . . . 159
7.21 NMEAData Class Reference . . . 159
7.21.1 Detailed Description . . . 160
7.21.2 Constructor & Destructor Documentation . . . 160
7.21.2.1 NMEAData() . . . 160
7.21.3 Member Function Documentation . . . 160
7.21.3.1 update_rmc_tokens() . . . 160
7.21.3.2 update_gga_tokens() . . . 160
7.21.3.3 get_rmc_tokens() . . . 160
7.21.3.4 get_gga_tokens() . . . 160

7.21.3.5 has_valid_time() . . . 161
7.21.3.6 get_unix_time() . . . 161
7.21.4 Member Data Documentation . . . 161
7.21.4.1 rmc_tokens_ . . . 161
7.21.4.2 gga_tokens_ . . . 161
7.21.4.3 rmc_mutex_ . . . 162
7.21.4.4 gga_mutex_ . . . 162
7.22 PowerManager Class Reference . . . 162
7.22.1 Detailed Description . . . 163
7.22.2 Constructor & Destructor Documentation . . . 163
7.22.2.1 PowerManager() . . . 163
7.22.3 Member Function Documentation . . . 164
7.22.3.1 initialize() . . . 164
7.22.3.2 read_device_ids() . . . 164
7.22.3.3 get_current_charge_solar() . . . 164
7.22.3.4 get_current_charge_usb() . . . 164
7.22.3.5 get_current_charge_total() . . . 165
7.22.3.6 get_current_draw() . . . 165
7.22.3.7 get_voltage_battery() . . . 165
7.22.3.8 get_voltage_5v() . . . 165
7.22.3.9 configure() . . . 165
7.22.3.10 is_charging_solar() . . . 165
7.22.3.11 is_charging_usb() . . . 166
7.22.3.12 check_power_alerts() . . . 166
7.22.4 Member Data Documentation . . . 166
7.22.4.1 SOLAR_CURRENT_THRESHOLD . . . 166
7.22.4.2 USB_CURRENT_THRESHOLD . . . 166
7.22.4.3 VOLTAGE_LOW_THRESHOLD . . . 167
7.22.4.4 VOLTAGE_OVERCHARGE_THRESHOLD . . . 167
7.22.4.5 FALL_RATE_THRESHOLD . . . 167
7.22.4.6 FALLING_TREND_REQUIRED . . . 167
7.22.4.7 ina3221_ . . . 167
7.22.4.8 initialized_ . . . 167
7.22.4.9 powerman_mutex_ . . . 167
7.22.4.10 charging_solar_active_ . . . 167
7.22.4.11 charging_usb_active_ . . . 168
7.23 SensorDataRecord Struct Reference . . . 168
7.23.1 Detailed Description . . . 168
7.23.2 Member Function Documentation . . . 168
7.23.2.1 to_csv() . . . 168
7.23.3 Member Data Documentation . . . 169
7.23.3.1 timestamp . . . 169

7.23.3.2 temperature . . . 169
7.23.3.3 pressure . . . 169
7.23.3.4 humidity . . . 169
7.23.3.5 light . . . 169
7.24 SensorWrapper Class Reference . . . 170
7.24.1 Detailed Description . . . 170
7.24.2 Constructor & Destructor Documentation . . . 171
7.24.2.1 SensorWrapper() . . . 171
7.24.3 Member Function Documentation . . . 171
7.24.3.1 get_instance() . . . 171
7.24.3.2 sensor_init() . . . 172
7.24.3.3 sensor_configure() . . . 172
7.24.3.4 sensor_read_data() . . . 173
7.24.3.5 get_sensor() . . . 174
7.24.3.6 scan_connected_sensors() . . . 174
7.24.3.7 get_available_sensors() . . . 174
7.24.4 Member Data Documentation . . . 175
7.24.4.1 sensors . . . 175
7.25 TelemetryRecord Struct Reference . . . 175
7.25.1 Detailed Description . . . 175
7.25.2 Member Function Documentation . . . 176
7.25.2.1 to_csv() . . . 176
7.25.3 Member Data Documentation . . . 176
7.25.3.1 timestamp . . . 176
7.25.3.2 build_version . . . 176
7.25.3.3 battery_voltage . . . 176
7.25.3.4 system_voltage . . . 176
7.25.3.5 charge_current_usb . . . 177
7.25.3.6 charge_current_solar . . . 177
7.25.3.7 discharge_current . . . 177
7.25.3.8 time . . . 177
7.25.3.9 latitude . . . 177
7.25.3.10 lat_dir . . . 177
7.25.3.11 longitude . . . 178
7.25.3.12 lon_dir . . . 178
7.25.3.13 speed . . . 178
7.25.3.14 course . . . 178
7.25.3.15 date . . . 178
7.25.3.16 fix_quality . . . 178
7.25.3.17 satellites . . . 179
7.25.3.18 altitude . . . 179

8 File Documentation 181
8.1 build_number.h File Reference . . . 181
8.1.1 Macro Definition Documentation . . . 181
8.1.1.1 BUILD_NUMBER . . . 181
8.2 build_number.h . . . 181
8.3 includes.h File Reference . . . 182
8.4 includes.h . . . 183
8.5 lib/clock/DS3231.cpp File Reference . . . 183
8.6 DS3231.cpp . . . 183
8.7 lib/clock/DS3231.h File Reference . . . 188
8.7.1 Macro Definition Documentation . . . 189
8.7.1.1 DS3231_DEVICE_ADRESS . . . 189
8.7.1.2 DS3231_SECONDS_REG . . . 189
8.7.1.3 DS3231_MINUTES_REG . . . 190
8.7.1.4 DS3231_HOURS_REG . . . 190
8.7.1.5 DS3231_DAY_REG . . . 190
8.7.1.6 DS3231_DATE_REG . . . 190
8.7.1.7 DS3231_MONTH_REG . . . 190
8.7.1.8 DS3231_YEAR_REG . . . 190
8.7.1.9 DS3231_CONTROL_REG . . . 191
8.7.1.10 DS3231_CONTROL_STATUS_REG . . . 191
8.7.1.11 DS3231_TEMPERATURE_MSB_REG . . . 191
8.7.1.12 DS3231_TEMPERATURE_LSB_REG . . . 191
8.7.2 Enumeration Type Documentation . . . 191
8.7.2.1 days_of_week . . . 191
8.8 DS3231.h . . . 192
8.9 lib/comms/commands/clock_commands.cpp File Reference . . . 193
8.9.1 Macro Definition Documentation . . . 194
8.9.1.1 CLOCK_GROUP . . . 194
8.9.1.2 TIME . . . 194
8.9.1.3 TIMEZONE_OFFSET . . . 194
8.9.1.4 CLOCK_SYNC_INTERVAL . . . 194
8.9.1.5 LAST_SYNC_TIME . . . 194
8.10 clock_commands.cpp . . . 195
8.11 lib/comms/commands/commands.cpp File Reference . . . 197
8.12 commands.cpp . . . 198
8.13 lib/comms/commands/commands.h File Reference . . . 199
8.14 commands.h . . . 201
8.15 lib/comms/commands/diagnostic_commands.cpp File Reference . . . 201
8.16 diagnostic_commands.cpp . . . 202
8.17 lib/comms/commands/event_commands.cpp File Reference . . . 204
8.18 event_commands.cpp . . . 204

8.19 lib/comms/commands/gps_commands.cpp File Reference . . . 205
8.19.1 Macro Definition Documentation . . . 206
8.19.1.1 GPS_GROUP . . . 206
8.19.1.2 POWER_STATUS_COMMAND . . . 206
8.19.1.3 PASSTHROUGH_COMMAND . . . 206
8.19.1.4 RMC_DATA_COMMAND . . . 206
8.19.1.5 GGA_DATA_COMMAND . . . 206
8.20 gps_commands.cpp . . . 207
8.21 lib/comms/commands/power_commands.cpp File Reference . . . 209
8.21.1 Macro Definition Documentation . . . 210
8.21.1.1 POWER_GROUP . . . 210
8.21.1.2 POWER_MANAGER_IDS . . . 210
8.21.1.3 VOLTAGE_BATTERY . . . 210
8.21.1.4 VOLTAGE_MAIN . . . 210
8.21.1.5 CHARGE_USB . . . 211
8.21.1.6 CHARGE_SOLAR . . . 211
8.21.1.7 CHARGE_TOTAL . . . 211
8.21.1.8 DRAW_TOTAL . . . 211
8.22 power_commands.cpp . . . 211
8.23 lib/comms/commands/sensor_commands.cpp File Reference . . . 213
8.23.1 Macro Definition Documentation . . . 214
8.23.1.1 SENSOR_GROUP . . . 214
8.23.1.2 SENSOR_READ . . . 214
8.23.1.3 SENSOR_CONFIGURE . . . 214
8.24 sensor_commands.cpp . . . 215
8.25 lib/comms/commands/storage_commands.cpp File Reference . . . 218
8.25.1 Macro Definition Documentation . . . 219
8.25.1.1 STORAGE_GROUP . . . 219
8.25.1.2 LIST_FILES_COMMAND . . . 219
8.25.1.3 MOUNT_COMMAND . . . 219
8.26 storage_commands.cpp . . . 220
8.27 lib/comms/commands/telemetry_commands.cpp File Reference . . . 221
8.27.1 Macro Definition Documentation . . . 222
8.27.1.1 TELEMETRY_GROUP . . . 222
8.27.1.2 GET_LAST_TELEMETRY_RECORD_COMMAND . . . 222
8.27.1.3 GET_LAST_SENSOR_RECORD_COMMAND . . . 222
8.27.2 Variable Documentation . . . 222
8.27.2.1 telemetry_mutex . . . 222
8.27.2.2 telemetry_buffer_count . . . 223
8.27.2.3 telemetry_buffer_write_index . . . 223
8.28 telemetry_commands.cpp . . . 223
8.29 lib/comms/communication.cpp File Reference . . . 224

8.29.1 Function Documentation . . . 224
8.29.1.1 initialize_radio() . . . 224
8.29.1.2 lora_tx_done_callback() . . . 225
8.29.2 Variable Documentation . . . 226
8.29.2.1 outgoing . . . 226
8.29.2.2 msgCount . . . 226
8.29.2.3 lastSendTime . . . 226
8.29.2.4 lastReceiveTime . . . 226
8.29.2.5 lastPrintTime . . . 226
8.29.2.6 interval . . . 226
8.30 communication.cpp . . . 227
8.31 lib/comms/communication.h File Reference . . . 227
8.31.1 Function Documentation . . . 228
8.31.1.1 initialize_radio() . . . 228
8.31.1.2 lora_tx_done_callback() . . . 229
8.31.1.3 on_receive() . . . 230
8.31.1.4 handle_uart_input() . . . 230
8.31.1.5 send_message() . . . 231
8.31.1.6 send_frame_uart() . . . 231
8.31.1.7 send_frame_lora() . . . 232
8.31.1.8 split_and_send_message() . . . 233
8.31.1.9 determine_unit() . . . 233
8.32 communication.h . . . 233
8.33 lib/comms/frame.cpp File Reference . . . 234
8.33.1 Detailed Description . . . 235
8.33.2 Typedef Documentation . . . 235
8.33.2.1 CommandHandler . . . 235
8.33.3 Variable Documentation . . . 235
8.33.3.1 eventRegister . . . 235
8.34 frame.cpp . . . 235
8.35 lib/comms/protocol.h File Reference . . . 237
8.35.1 Enumeration Type Documentation . . . 238
8.35.1.1 ErrorCode . . . 238
8.35.1.2 OperationType . . . 239
8.35.1.3 CommandAccessLevel . . . 239
8.35.1.4 ValueUnit . . . 239
8.35.1.5 ExceptionType . . . 240
8.35.1.6 Interface . . . 240
8.35.2 Function Documentation . . . 240
8.35.2.1 exception_type_to_string() . . . 240
8.35.2.2 error_code_to_string() . . . 241
8.35.2.3 operation_type_to_string() . . . 242

8.35.2.4 string_to_operation_type() . . . 243
8.35.2.5 hex_string_to_bytes() . . . 244
8.35.2.6 value_unit_type_to_string() . . . 244
8.35.3 Variable Documentation . . . 246
8.35.3.1 FRAME_BEGIN . . . 246
8.35.3.2 FRAME_END . . . 246
8.35.3.3 DELIMITER . . . 246
8.36 protocol.h . . . 246
8.37 lib/comms/receive.cpp File Reference . . . 247
8.37.1 Detailed Description . . . 248
8.37.2 Function Documentation . . . 248
8.37.2.1 on_receive() . . . 248
8.37.2.2 handle_uart_input() . . . 249
8.38 receive.cpp . . . 250
8.39 lib/comms/send.cpp File Reference . . . 251
8.39.1 Detailed Description . . . 251
8.39.2 Function Documentation . . . 251
8.39.2.1 send_message() . . . 251
8.39.2.2 send_frame_lora() . . . 252
8.39.2.3 send_frame_uart() . . . 252
8.39.2.4 split_and_send_message() . . . 252
8.40 send.cpp . . . 253
8.41 lib/comms/utils_converters.cpp File Reference . . . 254
8.41.1 Detailed Description . . . 254
8.41.2 Function Documentation . . . 254
8.41.2.1 exception_type_to_string() . . . 254
8.41.2.2 value_unit_type_to_string() . . . 255
8.41.2.3 operation_type_to_string() . . . 256
8.41.2.4 string_to_operation_type() . . . 257
8.41.2.5 error_code_to_string() . . . 258
8.41.2.6 hex_string_to_bytes() . . . 259
8.42 utils_converters.cpp . . . 260
8.43 lib/eventman/event_manager.cpp File Reference . . . 261
8.43.1 Detailed Description . . . 262
8.44 event_manager.cpp . . . 262
8.45 lib/eventman/event_manager.h File Reference . . . 263
8.45.1 Detailed Description . . . 265
8.45.2 Macro Definition Documentation . . . 265
8.45.2.1 EVENT_BUFFER_SIZE . . . 265
8.45.2.2 EVENT_FLUSH_THRESHOLD . . . 265
8.45.2.3 EVENT_LOG_FILE . . . 265
8.45.3 Function Documentation . . . 265

8.45.3.1 to_string() . . . 265
8.45.4 Variable Documentation . . . 265
8.45.4.1 id . . . 265
8.45.4.2 timestamp . . . 266
8.45.4.3 group . . . 266
8.45.4.4 event . . . 266
8.46 event_manager.h . . . 266
8.47 lib/location/gps_collector.cpp File Reference . . . 268
8.47.1 Macro Definition Documentation . . . 269
8.47.1.1 MAX_RAW_DATA_LENGTH . . . 269
8.47.2 Function Documentation . . . 269
8.47.2.1 splitString() . . . 269
8.47.2.2 collect_gps_data() . . . 270
8.47.3 Variable Documentation . . . 270
8.47.3.1 pause_gps_collection . . . 270
8.47.3.2 nmea_data . . . 270
8.48 gps_collector.cpp . . . 271
8.49 lib/location/gps_collector.h File Reference . . . 271
8.49.1 Function Documentation . . . 272
8.49.1.1 collect_gps_data() . . . 272
8.50 gps_collector.h . . . 273
8.51 lib/location/NMEA/NMEA_data.cpp File Reference . . . 273
8.51.1 Variable Documentation . . . 274
8.51.1.1 nmea_data . . . 274
8.52 NMEA_data.cpp . . . 274
8.53 lib/location/NMEA/NMEA_data.h File Reference . . . 275
8.53.1 Variable Documentation . . . 276
8.53.1.1 nmea_data . . . 276
8.54 NMEA_data.h . . . 276
8.55 lib/pin_config.cpp File Reference . . . 277
8.55.1 Variable Documentation . . . 277
8.55.1.1 lora_cs_pin . . . 277
8.55.1.2 lora_reset_pin . . . 277
8.55.1.3 lora_irq_pin . . . 277
8.55.1.4 lora_address_local . . . 278
8.55.1.5 lora_address_remote . . . 278
8.56 pin_config.cpp . . . 278
8.57 lib/pin_config.h File Reference . . . 278
8.57.1 Macro Definition Documentation . . . 279
8.57.1.1 DEBUG_UART_PORT . . . 279
8.57.1.2 DEBUG_UART_BAUD_RATE . . . 280
8.57.1.3 DEBUG_UART_TX_PIN . . . 280

8.57.1.4 DEBUG_UART_RX_PIN . . . 280
8.57.1.5 MAIN_I2C_PORT . . . 280
8.57.1.6 MAIN_I2C_SDA_PIN . . . 280
8.57.1.7 MAIN_I2C_SCL_PIN . . . 280
8.57.1.8 GPS_UART_PORT . . . 280
8.57.1.9 GPS_UART_BAUD_RATE . . . 280
8.57.1.10 GPS_UART_TX_PIN . . . 281
8.57.1.11 GPS_UART_RX_PIN . . . 281
8.57.1.12 GPS_POWER_ENABLE_PIN . . . 281
8.57.1.13 BUFFER_SIZE . . . 281
8.57.1.14 SD_SPI_PORT . . . 281
8.57.1.15 SD_MISO_PIN . . . 281
8.57.1.16 SD_MOSI_PIN . . . 281
8.57.1.17 SD_SCK_PIN . . . 281
8.57.1.18 SD_CS_PIN . . . 282
8.57.1.19 SD_CARD_DETECT_PIN . . . 282
8.57.1.20 SX1278_MISO . . . 282
8.57.1.21 SX1278_CS . . . 282
8.57.1.22 SX1278_SCK . . . 282
8.57.1.23 SX1278_MOSI . . . 282
8.57.1.24 SPI_PORT . . . 282
8.57.1.25 READ_BIT . . . 282
8.57.1.26 LORA_DEFAULT_SPI . . . 283
8.57.1.27 LORA_DEFAULT_SPI_FREQUENCY . . . 283
8.57.1.28 LORA_DEFAULT_SS_PIN . . . 283
8.57.1.29 LORA_DEFAULT_RESET_PIN . . . 283
8.57.1.30 LORA_DEFAULT_DIO0_PIN . . . 283
8.57.1.31 PA_OUTPUT_RFO_PIN . . . 283
8.57.1.32 PA_OUTPUT_PA_BOOST_PIN . . . 283
8.57.2 Variable Documentation . . . 284
8.57.2.1 lora_cs_pin . . . 284
8.57.2.2 lora_reset_pin . . . 284
8.57.2.3 lora_irq_pin . . . 284
8.57.2.4 lora_address_local . . . 284
8.57.2.5 lora_address_remote . . . 284
8.58 pin_config.h . . . 285
8.59 lib/powerman/INA3221/INA3221.cpp File Reference . . . 285
8.59.1 Detailed Description . . . 286
8.60 INA3221.cpp . . . 286
8.61 lib/powerman/INA3221/INA3221.h File Reference . . . 290
8.61.1 Detailed Description . . . 292
8.61.2 Enumeration Type Documentation . . . 292

8.61.2.1 ina3221_addr_t . . . 292
8.61.2.2 ina3221_ch_t . . . 292
8.61.2.3 ina3221_reg_t . . . 293
8.61.2.4 ina3221_conv_time_t . . . 293
8.61.2.5 ina3221_avg_mode_t . . . 294
8.61.3 Variable Documentation . . . 294
8.61.3.1 INA3221_CH_NUM . . . 294
8.61.3.2 SHUNT_VOLTAGE_LSB_UV . . . 294
8.62 INA3221.h . . . 295
8.63 lib/powerman/PowerManager.cpp File Reference . . . 297
8.64 PowerManager.cpp . . . 297
8.65 lib/powerman/PowerManager.h File Reference . . . 299
8.66 PowerManager.h . . . 300
8.67 lib/sensors/BH1750/BH1750.cpp File Reference . . . 301
8.68 BH1750.cpp . . . 301
8.69 lib/sensors/BH1750/BH1750.h File Reference . . . 302
8.69.1 Macro Definition Documentation . . . 303
8.69.1.1 _BH1750_DEVICE_ID . . . 303
8.69.1.2 _BH1750_MTREG_MIN . . . 303
8.69.1.3 _BH1750_MTREG_MAX . . . 304
8.69.1.4 _BH1750_DEFAULT_MTREG . . . 304
8.70 BH1750.h . . . 304
8.71 lib/sensors/BH1750/BH1750_WRAPPER.cpp File Reference . . . 304
8.72 BH1750_WRAPPER.cpp . . . 305
8.73 lib/sensors/BH1750/BH1750_WRAPPER.h File Reference . . . 306
8.74 BH1750_WRAPPER.h . . . 307
8.75 lib/sensors/BME280/BME280.cpp File Reference . . . 307
8.76 BME280.cpp . . . 308
8.77 lib/sensors/BME280/BME280.h File Reference . . . 311
8.78 BME280.h . . . 312
8.79 lib/sensors/BME280/BME280_WRAPPER.cpp File Reference . . . 314
8.80 BME280_WRAPPER.cpp . . . 314
8.81 lib/sensors/BME280/BME280_WRAPPER.h File Reference . . . 315
8.82 BME280_WRAPPER.h . . . 315
8.83 lib/sensors/HMC5883L/HMC5883L.cpp File Reference . . . 316
8.84 HMC5883L.cpp . . . 316
8.85 lib/sensors/HMC5883L/HMC5883L.h File Reference . . . 317
8.86 HMC5883L.h . . . 318
8.87 lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp File Reference . . . 318
8.88 HMC5883L_WRAPPER.cpp . . . 319
8.89 lib/sensors/HMC5883L/HMC5883L_WRAPPER.h File Reference . . . 320
8.90 HMC5883L_WRAPPER.h . . . 321

8.91 lib/sensors/ISensor.cpp File Reference . . . 321
8.91.1 Detailed Description . . . 321
8.92 ISensor.cpp . . . 322
8.93 lib/sensors/ISensor.h File Reference . . . 323
8.93.1 Enumeration Type Documentation . . . 324
8.93.1.1 SensorType . . . 324
8.93.1.2 SensorDataTypeIdentifier . . . 324
8.94 ISensor.h . . . 325
8.95 lib/sensors/MPU6050/MPU6050.cpp File Reference . . . 325
8.96 MPU6050.cpp . . . 325
8.97 lib/sensors/MPU6050/MPU6050.h File Reference . . . 326
8.98 MPU6050.h . . . 326
8.99 lib/sensors/MPU6050/MPU6050_WRAPPER.cpp File Reference . . . 326
8.100 MPU6050_WRAPPER.cpp . . . 326
8.101 lib/sensors/MPU6050/MPU6050_WRAPPER.h File Reference . . . 326
8.102 MPU6050_WRAPPER.h . . . 327
8.103 lib/storage/storage.cpp File Reference . . . 328
8.103.1 Detailed Description . . . 328
8.103.2 Function Documentation . . . 329
8.103.2.1 fs_init() . . . 329
8.103.3 Variable Documentation . . . 330
8.103.3.1 sd_card_mounted . . . 330
8.104 storage.cpp . . . 330
8.105 lib/storage/storage.h File Reference . . . 330
8.105.1 Function Documentation . . . 331
8.105.1.1 fs_init() . . . 331
8.105.1.2 fs_open_file() . . . 332
8.105.1.3 fs_write_file() . . . 332
8.105.1.4 fs_read_file() . . . 333
8.105.1.5 fs_close_file() . . . 333
8.105.1.6 fs_file_exists() . . . 333
8.105.2 Variable Documentation . . . 333
8.105.2.1 sd_card_mounted . . . 333
8.106 storage.h . . . 333
8.107 lib/telemetry/telemetry_manager.cpp File Reference . . . 334
8.107.1 Detailed Description . . . 336
8.107.2 Macro Definition Documentation . . . 336
8.107.2.1 TELEMETRY_CSV_PATH . . . 336
8.107.2.2 SENSOR_DATA_CSV_PATH . . . 336
8.107.2.3 DEFAULT_SAMPLE_INTERVAL_MS . . . 336
8.107.2.4 DEFAULT_FLUSH_THRESHOLD . . . 336
8.107.3 Variable Documentation . . . 336

8.107.3.1 powerManager . . . 336
8.107.3.2 systemClock . . . 337
8.107.3.3 nmea_data . . . 337
8.107.3.4 sample_interval_ms . . . 337
8.107.3.5 flush_threshold . . . 337
8.108 telemetry_manager.cpp . . . 337
8.109 lib/telemetry/telemetry_manager.h File Reference . . . 341
8.109.1 Detailed Description . . . 343
8.110 telemetry_manager.h . . . 343
8.111 lib/utils.cpp File Reference . . . 345
8.111.1 Detailed Description . . . 346
8.111.2 Function Documentation . . . 346
8.111.2.1 get_level_color() . . . 346
8.111.2.2 get_level_prefix() . . . 347
8.111.2.3 uart_print() . . . 348
8.111.2.4 base64_encode() . . . 350
8.111.2.5 calculate_checksum() . . . 351
8.111.3 Variable Documentation . . . 351
8.111.3.1 uart_mutex . . . 351
8.111.3.2 g_uart_verbosity . . . 351
8.111.3.3 base64_chars . . . 351
8.112 utils.cpp . . . 352
8.113 lib/utils.h File Reference . . . 353
8.113.1 Detailed Description . . . 354
8.113.2 Macro Definition Documentation . . . 355
8.113.2.1 ANSI_RED . . . 355
8.113.2.2 ANSI_GREEN . . . 355
8.113.2.3 ANSI_YELLOW . . . 355
8.113.2.4 ANSI_BLUE . . . 355
8.113.2.5 ANSI_CYAN . . . 355
8.113.2.6 ANSI_RESET . . . 355
8.113.3 Enumeration Type Documentation . . . 355
8.113.3.1 VerbosityLevel . . . 355
8.113.4 Function Documentation . . . 356
8.113.4.1 uart_print() . . . 356
8.113.4.2 base64_encode() . . . 358
8.113.4.3 calculate_checksum() . . . 359
8.113.5 Variable Documentation . . . 359
8.113.5.1 g_uart_verbosity . . . 359
8.114 utils.h . . . 359
8.115 main.cpp File Reference . . . 360
8.115.1 Macro Definition Documentation . . . 361

8.115.1.1 LOG_FILENAME . . . 361
8.115.2 Function Documentation . . . 361
8.115.2.1 core1_entry() . . . 361
8.115.2.2 init_pico_hw() . . . 362
8.115.2.3 init_modules() . . . 362
8.115.2.4 main() . . . 363
8.115.3 Variable Documentation . . . 363
8.115.3.1 powerManager . . . 363
8.115.3.2 systemClock . . . 364
8.115.3.3 pause_gps_collection . . . 364
8.115.3.4 buffer . . . 364
8.115.3.5 buffer_index . . . 364
8.116 main.cpp . . . 364
8.117 test/comms/test_comand_handlers.cpp File Reference . . . 366
8.117.1 Function Documentation . . . 367
8.117.1.1 send_frame_uart() . . . 367
8.117.1.2 send_frame_lora() . . . 368
8.117.1.3 setUp() . . . 368
8.117.1.4 tearDown() . . . 369
8.117.1.5 test_command_handler_get_operation() . . . 369
8.117.1.6 test_command_handler_set_operation() . . . 369
8.117.1.7 test_command_handler_invalid_operation() . . . 370
8.117.2 Variable Documentation . . . 370
8.117.2.1 uart_send_called . . . 370
8.117.2.2 lora_send_called . . . 371
8.117.2.3 last_frame_sent . . . 371
8.118 test_comand_handlers.cpp . . . 371
8.119 test/comms/test_converters.cpp File Reference . . . 372
8.119.1 Function Documentation . . . 372
8.119.1.1 test_operation_type_conversion() . . . 372
8.119.1.2 test_value_unit_type_conversion() . . . 373
8.119.1.3 test_exception_type_conversion() . . . 373
8.119.1.4 test_hex_string_conversion() . . . 374
8.120 test_converters.cpp . . . 374
8.121 test/comms/test_error_codes.cpp File Reference . . . 375
8.121.1 Function Documentation . . . 375
8.121.1.1 setUp() . . . 375
8.121.1.2 tearDown() . . . 376
8.121.1.3 test_error_code_conversion() . . . 376
8.122 test_error_codes.cpp . . . 376
8.123 test/comms/test_frame_build.cpp File Reference . . . 377
8.123.1 Function Documentation . . . 377

8.123.1.1 test_frame_build_success() . . . 377
8.123.1.2 test_frame_build_error() . . . 378
8.123.1.3 test_frame_build_info() . . . 378
8.124 test_frame_build.cpp . . . 379
8.125 test/comms/test_frame_coding.cpp File Reference . . . 379
8.125.1 Function Documentation . . . 380
8.125.1.1 test_frame_encode_basic() . . . 380
8.125.1.2 test_frame_decode_basic() . . . 380
8.125.1.3 test_frame_decode_invalid_header() . . . 381
8.126 test_frame_coding.cpp . . . 381
8.127 test/comms/test_frame_common.h File Reference . . . 382
8.127.1 Function Documentation . . . 383
8.127.1.1 create_test_frame() . . . 383
8.128 test_frame_common.h . . . 383
8.129 test/comms/test_frame_send.cpp File Reference . . . 384
8.129.1 Function Documentation . . . 384
8.129.1.1 setUp() . . . 384
8.129.1.2 tearDown() . . . 384
8.129.1.3 test_send_frame_uart() . . . 384
8.130 test_frame_send.cpp . . . 385
8.131 test/mocks/hardware_mocks.cpp File Reference . . . 385
8.131.1 Function Documentation . . . 386
8.131.1.1 mock_uart_puts() . . . 386
8.131.1.2 mock_uart_init() . . . 386
8.131.1.3 mock_spi_write_blocking() . . . 386
8.131.1.4 mock_spi_read_blocking() . . . 386
8.131.2 Variable Documentation . . . 387
8.131.2.1 mock_uart_enabled . . . 387
8.131.2.2 uart_output_buffer . . . 387
8.131.2.3 mock_spi_enabled . . . 387
8.131.2.4 spi_output_buffer . . . 387
8.132 hardware_mocks.cpp . . . 387
8.133 test/mocks/hardware_mocks.h File Reference . . . 388
8.133.1 Function Documentation . . . 389
8.133.1.1 mock_uart_puts() . . . 389
8.133.1.2 mock_uart_init() . . . 389
8.133.1.3 mock_spi_write_blocking() . . . 389
8.133.1.4 mock_spi_read_blocking() . . . 389
8.133.2 Variable Documentation . . . 389
8.133.2.1 mock_uart_enabled . . . 389
8.133.2.2 uart_output_buffer . . . 389
8.133.2.3 mock_spi_enabled . . . 390

8.133.2.4 spi_output_buffer . . . 390
8.134 hardware_mocks.h . . . 390
8.135 test/test_runner.cpp File Reference . . . 390
8.135.1 Function Documentation . . . 391
8.135.1.1 test_frame_encode_basic() . . . 391
8.135.1.2 test_frame_decode_basic() . . . 392
8.135.1.3 test_frame_decode_invalid_header() . . . 392
8.135.1.4 test_frame_build_success() . . . 393
8.135.1.5 test_frame_build_error() . . . 393
8.135.1.6 test_frame_build_info() . . . 394
8.135.1.7 test_operation_type_conversion() . . . 395
8.135.1.8 test_value_unit_type_conversion() . . . 395
8.135.1.9 test_exception_type_conversion() . . . 396
8.135.1.10 test_hex_string_conversion() . . . 397
8.135.1.11 test_command_handler_get_operation() . . . 397
8.135.1.12 test_command_handler_set_operation() . . . 398
8.135.1.13 test_command_handler_invalid_operation() . . . 399
8.135.1.14 test_error_code_conversion() . . . 399
8.135.1.15 main() . . . 400
8.136 test_runner.cpp . . . 400

Index **403**

Chapter 1

Clock Commands

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

Command ID: 3.3

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

Command ID: 7.2

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

Command ID: 2

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

Command ID: 1

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

Command ID: 0

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

Command ID: 2.5

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

Command ID: 2.6

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

Command ID: 2.4

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

Command ID: 2.7

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

Command ID: 5.2

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

Command ID: 7.4

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

Command ID: 5.1

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

Command ID: 3.7

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

Command ID: 8.2

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

Command ID: 2.0

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

Command ID: 7.3

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

Command ID: 3.0

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

Command ID: 4.2

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

Command ID: 2.3

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

Command ID: 2.2

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

Command ID: 7.1

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

Command ID: 6.0

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

Command ID: 6.4

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

Command ID: 3.1

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

Command ID: 3.0

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

Command ID: 3.1

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

Command ID: 1.8

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Clock Management Commands 11
Command System 15
Diagnostic Commands 17
Event Commands 21
GPS Commands 24
Power Commands 28
Sensor Commands 34
Storage Commands 37
Telemetry Buffer Commands 39
Frame Handling 41
Event Manager 46
INA3221 Power Monitor 53
- Configuration Functions 53
- Measurement Functions 63
- Alert Functions 65

Telemetry Manager 71

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

BH1750 81
BME280 87
BME280CalibParam 96
INA3221::conf_reg_t 102
DS3231 104
ds3231_data_t 123
EventEmitter 125
EventLog 127
EventManager 129
  EventManagerImpl 134
FileHandle 137
Frame 138
HMC5883L 140
INA3221 146
ISensor 152
  BH1750Wrapper 84
  BME280Wrapper 100
  HMC5883LWrapper 143
  MPU6050Wrapper 157
INA3221::masken_reg_t 154
NMEAData 159
PowerManager 162
SensorDataRecord 168
SensorWrapper 170
TelemetryRecord 175

Chapter 4

Class Index

4.1 Class List

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

BH1750 81
BH1750Wrapper 84
BME280 87
BME280CalibParam 96
BME280Wrapper 100
INA3221::conf_reg_t
Configuration register bit fields 102
DS3231
Class for interfacing with the DS3231 real-time clock 104
ds3231_data_t
Structure to hold time and date information from DS3231 123
EventEmitter
Provides a static method for emitting events 125
EventLog
Represents a single event log entry 127
EventManager
Manages the event logging system 129
EventManagerImpl
Implementation of the EventManager class 134
FileHandle 137
Frame
Represents a communication frame used for data exchange 138
HMC5883L 140
HMC5883LWrapper 143
INA3221
INA3221 Triple-Channel Power Monitor driver class 146
ISensor 152
INA3221::masken_reg_t
Mask/Enable register bit fields 154
MPU6050Wrapper 157
NMEAData 159
PowerManager 162
SensorDataRecord 168
SensorWrapper
Manages different sensor types and provides a unified interface for accessing sensor data . . 170
TelemetryRecord
Structure representing a single telemetry data point 175

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

build_number.h . . . 181
includes.h . . . 182
main.cpp . . . 360
lib/pin_config.cpp . . . 277
lib/pin_config.h . . . 278
lib/utils.cpp
Implementation of utility functions for the Kubisat firmware . . . 345
lib/utils.h
Utility functions and definitions for the Kubisat firmware . . . 353
lib/clock/DS3231.cpp . . . 183
lib/clock/DS3231.h . . . 188
lib/comms/communication.cpp . . . 224
lib/comms/communication.h . . . 227
lib/comms/frame.cpp
Implements functions for encoding, decoding, building, and processing Frames . . . 234
lib/comms/protocol.h . . . 237
lib/comms/receive.cpp
Implements functions for receiving and processing data, including LoRa and UART input . . . 247
lib/comms/send.cpp
Implements functions for sending data, including LoRa messages and Frames . . . 251
lib/comms/utils_converters.cpp
Implements utility functions for converting between different data types . . . 254
lib/comms/commands/clock_commands.cpp . . . 193
lib/comms/commands/commands.cpp . . . 197
lib/comms/commands/commands.h . . . 199
lib/comms/commands/diagnostic_commands.cpp . . . 201
lib/comms/commands/event_commands.cpp . . . 204
lib/comms/commands/gps_commands.cpp . . . 205
lib/comms/commands/power_commands.cpp . . . 209
lib/comms/commands/sensor_commands.cpp . . . 213
lib/comms/commands/storage_commands.cpp . . . 218
lib/comms/commands/telemetry_commands.cpp . . . 221
lib/eventman/event_manager.cpp
Implements the event management system for the Kubisat firmware . . . 261
lib/eventman/event_manager.h
Manages the event logging system for the Kubisat firmware . . . 263

lib/location/gps_collector.cpp . . . 268
lib/location/gps_collector.h . . . 271
lib/location/NMEA/NMEA_data.cpp . . . 273
lib/location/NMEA/NMEA_data.h . . . 275
lib/powerman/PowerManager.cpp . . . 297
lib/powerman/PowerManager.h . . . 299
lib/powerman/INA3221/INA3221.cpp
Implementation of the INA3221 power monitor driver . . . 285
lib/powerman/INA3221/INA3221.h
Header file for the INA3221 triple-channel power monitor driver . . . 290
lib/sensors/ISensor.cpp
Implements the SensorWrapper class for managing different sensor types . . . 321
lib/sensors/ISensor.h . . . 323
lib/sensors/BH1750/BH1750.cpp . . . 301
lib/sensors/BH1750/BH1750.h . . . 302
lib/sensors/BH1750/BH1750_WRAPPER.cpp . . . 304
lib/sensors/BH1750/BH1750_WRAPPER.h . . . 306
lib/sensors/BME280/BME280.cpp . . . 307
lib/sensors/BME280/BME280.h . . . 311
lib/sensors/BME280/BME280_WRAPPER.cpp . . . 314
lib/sensors/BME280/BME280_WRAPPER.h . . . 315
lib/sensors/HMC5883L/HMC5883L.cpp . . . 316
lib/sensors/HMC5883L/HMC5883L.h . . . 317
lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp . . . 318
lib/sensors/HMC5883L/HMC5883L_WRAPPER.h . . . 320
lib/sensors/MPU6050/MPU6050.cpp . . . 325
lib/sensors/MPU6050/MPU6050.h . . . 326
lib/sensors/MPU6050/MPU6050_WRAPPER.cpp . . . 326
lib/sensors/MPU6050/MPU6050_WRAPPER.h . . . 326
lib/storage/storage.cpp
Implements file system operations for the Kubisat firmware . . . 328
lib/storage/storage.h . . . 330
lib/telemetry/telemetry_manager.cpp
Implementation of telemetry collection and storage functionality . . . 334
lib/telemetry/telemetry_manager.h
System telemetry collection and logging . . . 341
test/test_runner.cpp . . . 390
test/comms/test_comand_handlers.cpp . . . 366
test/comms/test_converters.cpp . . . 372
test/comms/test_error_codes.cpp . . . 375
test/comms/test_frame_build.cpp . . . 377
test/comms/test_frame_coding.cpp . . . 379
test/comms/test_frame_common.h . . . 382
test/comms/test_frame_send.cpp . . . 384
test/mocks/hardware_mocks.cpp . . . 385
test/mocks/hardware_mocks.h . . . 388

Chapter 6

Topic Documentation

6.1 Clock Management Commands

Commands for managing system time and clock settings.

Functions

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

  Handler for getting and setting system time.
- std::vector< Frame > handle_timezone_offset (const std::string &param, OperationType operationType)

  Handler for getting and setting timezone offset.
- std::vector< Frame > handle_clock_sync_interval (const std::string &param, OperationType operationType)

  Handler for getting and setting clock synchronization interval.
- std::vector< Frame > handle_get_last_sync_time (const std::string &param, OperationType operationType)

  Handler for getting last clock sync time.

Variables

- DS3231 systemClock

6.1.1 Detailed Description

Commands for managing system time and clock settings.

6.1.2 Function Documentation

6.1.2.1 handle_time()

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

Handler for getting and setting system time.

Parameters

param	For SET: Unix timestamp as string, for GET: empty string
operationType	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

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

Returns

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

Note

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

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

Command Command ID: 3.1

Definition at line 97 of file clock_commands.cpp.

Here is the call graph for this function:

6.1.2.3 handle_clock_sync_interval()

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

Handler for getting and setting clock synchronization interval.

Parameters

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

Returns

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

Note

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

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

Command Command ID: 3.3

Definition at line 158 of file clock_commands.cpp.

Here is the call graph for this function:

6.1.2.4 handle_get_last_sync_time()

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

Handler for getting last clock sync time.

Parameters

param	Empty string expected
operationType	GET

Returns

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

Note

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

Command Command ID: 3.7

Definition at line 216 of file clock_commands.cpp.

Here is the call graph for this function:

6.1.3 Variable Documentation

6.1.3.1 systemClock

```
DS3231 systemClock [extern]
```

6.2 Command System

Core command system implementation.

Typedefs

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

  Function type for command handlers.
- using CommandMap = std::map<uint32_t, CommandHandler>

  Map type for storing command handlers.

Functions

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

  Executes a command based on its key.

Variables

- CommandMap command_handlers

  Global map of all command handlers.

6.2.1 Detailed Description

Core command system implementation.

6.2.2 Typedef Documentation

6.2.2.1 CommandHandler

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

Function type for command handlers.

Definition at line 15 of file commands.cpp.

6.2.2.2 CommandMap

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

Map type for storing command handlers.

Definition at line 21 of file commands.cpp.

6.2.3 Function Documentation

6.2.3.1 execute_command()

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

Executes a command based on its key.

Parameters

commandKey	Combined group and command ID (group << 8 | command)
param	Command parameter string
operationType	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 67 of file commands.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.2.4 Variable Documentation

6.2.4.1 command_handlers

```
CommandMap command_handlers
```

Initial value:

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

Global map of all command handlers.

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

Definition at line 27 of file commands.cpp.

6.3 Diagnostic Commands

Functions

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

  Handler for listing all available commands on UART.
- std::vector< Frame > handle_get_build_version (const std::string &param, OperationType operationType)

  Get firmware build version.
- std::vector< Frame > handle_verbosity (const std::string &param, OperationType operationType)

  Handles setting or getting the UART verbosity level.
- std::vector< Frame > handle_enter_bootloader_mode (const std::string &param, OperationType operation↩ Type)

  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

param	Empty string expected
operationType	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

param	Empty string expected
operationType	GET

Returns

One-element vector with result frame

Note

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

Get the firmware build version

Command Command ID: 1

Definition at line 77 of file diagnostic_commands.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.3.2.3 handle_verbosity()

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

Handles setting or getting the UART verbosity level.

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

Parameters

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

Returns

Vector containing one frame indicating the result of the operation.

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

Note

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

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

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

Command Command ID: 1.8

Definition at line 119 of file diagnostic_commands.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.3.2.4 handle_enter_bootloader_mode()

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

Reboot system to USB firmware loader.

Parameters

param	Empty string expected
operationType	Must be SET

Returns

Frame with operation result

Note

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

Reboot the system to USB firmware loader

Command Command ID: 2

Definition at line 158 of file diagnostic_commands.cpp.

Here is the call graph for this function:

6.3.3 Variable Documentation

6.3.3.1 g_pending_bootloader_reset

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

Definition at line 7 of file main.cpp.

6.4 Event Commands

Commands for accessing and managing system event logs.

Functions

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

  Handler for retrieving last N events from the event log.
- std::vector< Frame > handle_get_event_count (const std::string &param, OperationType operationType)

  Handler for getting total number of events in the log.

6.4.1 Detailed Description

Commands for accessing and managing system event logs.

6.4.2 Function Documentation

6.4.2.1 handle_get_last_events()

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

Handler for retrieving last N events from the event log.

Parameters

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

Returns

Frame containing:

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

Note

KBST;0;GET;5;1;[N];TSBK - Retrieves the last N events. If N is 0, retrieves all events.

Returns up to 10 most recent events per frame.

Command Command ID: 5.1

Definition at line 33 of file event_commands.cpp.

Here is the call graph for this function:

6.4.2.2 handle_get_event_count()

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

Handler for getting total number of events in the log.

Parameters

param	Empty string expected
operationType	GET

Returns

Frame containing:

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

Note

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

Returns the total number of events in the log

Command Command ID: 5.2

Definition at line 100 of file event_commands.cpp.

Here is the call graph for this function:

6.5 GPS Commands

Commands for controlling and monitoring the GPS module.

Functions

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

  Handler for controlling GPS module power state.
- std::vector< Frame > handle_enable_gps_uart_passthrough (const std::string &param, OperationType operationType)

  Handler for enabling GPS transparent mode (UART pass-through)
- std::vector< Frame > handle_get_rmc_data (const std::string &param, OperationType operationType)

  Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- std::vector< Frame > handle_get_gga_data (const std::string &param, OperationType operationType)

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

6.5.1 Detailed Description

Commands for controlling and monitoring the GPS module.

6.5.2 Function Documentation

6.5.2.1 handle_gps_power_status()

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

Handler for controlling GPS module power state.

Parameters

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

Returns

Vector of Frames containing:

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

Note

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

Return current GPS module power state: ON/OFF

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

POWER - 0 - OFF, 1 - ON

Command Command ID: 7.1

Definition at line 32 of file gps_commands.cpp.

Here is the call graph for this function:

6.5.2.2 handle_enable_gps_uart_passthrough()

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

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

Parameters

param	TIMEOUT in seconds (optional, defaults to 60)
operationType	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 89 of file gps_commands.cpp.

Here is the call graph for this function:

6.5.2.3 handle_get_rmc_data()

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

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

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

Command Command ID: 7.3

Definition at line 193 of file gps_commands.cpp.

Here is the call graph for this function:

6.5.2.4 handle_get_gga_data()

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

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

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

Command Command ID: 7.4

Definition at line 235 of file gps_commands.cpp.

Here is the call graph for this function:

6.6 Power Commands

Commands for monitoring power subsystem and battery management.

Functions

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

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

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

  Handler for getting 5V rail voltage.
- std::vector< Frame > handle_get_current_charge_usb (const std::string &param, OperationType operation↩ Type)

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

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

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

  Handler for getting system current draw.

6.6.1 Detailed Description

Commands for monitoring power subsystem and battery management.

6.6.2 Function Documentation

6.6.2.1 handle_get_power_manager_ids()

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

Handler for retrieving Power Manager IDs.

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

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

Command Command ID: 2.0

Definition at line 30 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.2 handle_get_voltage_battery()

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

Handler for getting battery voltage.

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

This command is used to retrieve the battery voltage

Command Command ID: 2.2

Definition at line 64 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.3 handle_get_voltage_5v()

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

Handler for getting 5V rail voltage.

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

This command is used to retrieve the 5V rail voltage

Command Command ID: 2.3

Definition at line 98 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.4 handle_get_current_charge_usb()

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

Handler for getting USB charge current.

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

This command is used to retrieve the USB charge current

Command Command ID: 2.4

Definition at line 132 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.5 handle_get_current_charge_solar()

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

Handler for getting solar panel charge current.

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

This command is used to retrieve the solar panel charge current

Command Command ID: 2.5

Definition at line 166 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.6 handle_get_current_charge_total()

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

Handler for getting total charge current.

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

This command is used to retrieve the total charge current

Command Command ID: 2.6

Definition at line 200 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.7 handle_get_current_draw()

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

Handler for getting system current draw.

Parameters

param	Empty string expected
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

This command is used to retrieve the system current draw

Command Command ID: 2.7

Definition at line 234 of file power_commands.cpp.

Here is the call graph for this function:

6.7 Sensor Commands

Commands for reading and configuring sensors.

Functions

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

6.7.1 Detailed Description

Commands for reading and configuring sensors.

6.7.2 Function Documentation

6.7.2.1 handle_get_sensor_data()

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

Handler for reading sensor data.

Parameters

param	String in format "sensor_type[-data_type]" where: • sensor_type: "light", "environment", "magnetometer", "imu" • data_type (optional): specific data type for the sensor – For light: "light_level" – For environment: "temperature", "pressure", "humidity" – For magnetometer: "mag_field_x", "mag_field_y", "mag_field_z" – For IMU: "gyro_x", "gyro_y", "gyro_z", "accel_x", "accel_y", "accel_z"
operationType	GET

Returns

Vector of Frames containing:

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

Note

KBST;0;GET;3;0;light-light_level;TSBK

This command is used to read data from sensors

Command Command ID: 3.0

Definition at line 36 of file sensor_commands.cpp.

Here is the call graph for this function:

6.7.2.2 handle_sensor_config()

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

Handler for configuring sensors.

Parameters

param	String in format "sensor_type;key1:value1|key2:value2|..." • sensor_type: "light", "environment", "magnetometer", "imu" • key-value pairs for configuration parameters
operationType	SET

Returns

Vector of Frames containing:

- Success: Success message
- Error: Error message

Note

KBST;0;SET;3;1;light;measurement_mode:continuously_high_resolution;TSBK

This command is used to configure sensors

Command Command ID: 3.1

Definition at line 236 of file sensor_commands.cpp.

Here is the call graph for this function:

6.7.2.3 handle_get_sensor_list()

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

Handler for listing available sensors.

Parameters

param	Empty string or optional filter criteria
operationType	GET

Returns

Vector of Frames containing:

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

Note

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

This command is used to get a list of available sensors

Command Command ID: 4.2

Definition at line 320 of file sensor_commands.cpp.

Here is the call graph for this function:

6.8 Storage Commands

Commands for interacting with the SD card storage.

Functions

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

  Handles the list files command.
- std::vector< Frame > handle_mount (const std::string &param, OperationType operationType)

  Handles the SD card mount/unmount command.

6.8.1 Detailed Description

Commands for interacting with the SD card storage.

6.8.2 Function Documentation

6.8.2.1 handle_list_files()

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

Handles the list files command.

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

Parameters

param	Unused.
operationType	The operation type (must be GET).

Returns

A vector of Frames indicating the result of the operation.

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

Note

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

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

Command Command ID: 6.0

Definition at line 37 of file storage_commands.cpp.

Here is the call graph for this function:

6.8.2.2 handle_mount()

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

Handles the SD card mount/unmount command.

This function mounts or unmounts the SD card.

Parameters

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

Returns

A vector of Frames indicating the result of the operation.

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

Note

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

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

Command Command ID: 6.4

Definition at line 123 of file storage_commands.cpp.

Here is the call graph for this function:

6.9 Telemetry Buffer Commands

Commands for interacting with the telemetry buffer.

Functions

- std::vector< Frame > handle_get_last_telemetry_record (const std::string &param, OperationType operationType)

  Handles the get last record command.
- std::vector< Frame > handle_get_last_sensor_record (const std::string &param, OperationType operation↩ Type)

  Handles the get last sensor record command.

6.9.1 Detailed Description

Commands for interacting with the telemetry buffer.

6.9.2 Function Documentation

6.9.2.1 handle_get_last_telemetry_record()

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

Handles the get last record command.

This function reads the last record from the telemetry buffer, base64 encodes it, and sends the encoded data as a response.

Parameters

param	Unused.
operationType	The operation type (must be GET).

Returns

A vector of Frames indicating the result of the operation.

- Success: Frame with base64 encoded telemetry data.
- Error: Frame with error message (e.g., "No telemetry data available").

Note

KBST;0;GET;8;2;;TSBK

This command retrieves the last telemetry record from the buffer and sends it base64 encoded.

Command Command ID: 8.2

Definition at line 37 of file telemetry_commands.cpp.

Here is the call graph for this function:

6.9.2.2 handle_get_last_sensor_record()

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

Handles the get last sensor record command.

This function retrieves the last sensor record from the telemetry manager, and sends the data as a response.

Parameters

param	Unused.
operationType	The operation type (must be GET).

Returns

A vector of Frames indicating the result of the operation.

Definition at line 71 of file telemetry_commands.cpp.

Here is the call graph for this function:

6.10 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.

6.10.1 Detailed Description

Functions for encoding, decoding and building communication frames.

6.10.2 Function Documentation

6.10.2.1 frame_encode()

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

Encodes a Frame instance into a string.

Parameters

frame	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:

6.10.2.2 frame_decode()

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

Decodes a string into a Frame instance.

Parameters

encodedFrame	The string to decode.

Returns

The Frame instance decoded from the string.

Exceptions

std::runtime_error	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:

6.10.2.3 frame_process()

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

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

Parameters

data	The Frame data in string format.

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

Definition at line 117 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.10.2.4 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

result	The execution result.
group	The group ID.
command	The command ID within the group.
value	The payload value.
unit	The unit of measurement for the payload value.

Returns

The Frame instance.

Definition at line 158 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.11 Event Manager

Classes and functions for managing event logging.

Files

- file event_manager.cpp

  Implements the event management system for the Kubisat firmware.

Classes

- class EventLog

  Represents a single event log entry.
- class EventManager

  Manages the event logging system.
- class EventManagerImpl

  Implementation of the EventManager class.
- class EventEmitter

  Provides a static method for emitting events.

Enumerations

- enum class EventGroup : uint8_t { EventGroup::SYSTEM = 0x00 , EventGroup::POWER = 0x01 , EventGroup::COMMS = 0x02 , EventGroup::GPS = 0x03 , EventGroup::CLOCK = 0x04 }

  Represents the group to which an event belongs.
- enum class SystemEvent : uint8_t { SystemEvent::BOOT = 0x01 , SystemEvent::SHUTDOWN = 0x02 , SystemEvent::WATCHDOG_RESET = 0x03 , SystemEvent::CORE1_START = 0x04 , SystemEvent::CORE1_STOP = 0x05 }

  Represents specific system events.
- enum class PowerEvent : uint8_t { PowerEvent::LOW_BATTERY = 0x01 , PowerEvent::OVERCHARGE = 0x02 , PowerEvent::POWER_FALLING = 0x03 , PowerEvent::POWER_NORMAL = 0x04 , PowerEvent::SOLAR_ACTIVE = 0x05 , PowerEvent::SOLAR_INACTIVE = 0x06 , PowerEvent::USB_CONNECTED = 0x07 , PowerEvent::USB_DISCONNECTED = 0x08 }

  Represents specific power-related events.
- enum class CommsEvent : uint8_t { CommsEvent::RADIO_INIT = 0x01 , CommsEvent::RADIO_ERROR = 0x02 , CommsEvent::MSG_RECEIVED = 0x03 , CommsEvent::MSG_SENT = 0x04 , CommsEvent::UART_ERROR = 0x06 }

  Represents specific communication-related events.
- enum class GPSEvent : uint8_t { GPSEvent::LOCK = 0x01 , GPSEvent::LOST = 0x02 , GPSEvent::ERROR = 0x03 , GPSEvent::POWER_ON = 0x04 , GPSEvent::POWER_OFF = 0x05 , GPSEvent::DATA_READY = 0x06 , GPSEvent::PASS_THROUGH_START = 0x07 , GPSEvent::PASS_THROUGH_END = 0x08 }

  Represents specific GPS-related events.
- enum class ClockEvent : uint8_t { ClockEvent::CHANGED = 0x01 , ClockEvent::GPS_SYNC = 0x02 , ClockEvent::GPS_SYNC_DATA_NOT_READY = 0x03 }

  Represents specific clock-related events.

Functions

- class EventLog __attribute__ ((packed))
- void EventManager::log_event (uint8_t group, uint8_t event)

  Logs an event.
- const EventLog & EventManager::get_event (size_t index) const

  Retrieves an event from the event buffer.

Variables

- volatile uint16_t eventLogId = 0

  Global event log ID counter.
- DS3231 systemClock

  External declaration of the system clock.
- EventManagerImpl eventManager

  Global instance of the EventManager implementation.
- class EventManager __attribute__
- EventManagerImpl eventManager

  Global instance of the EventManagerImpl class.

6.11.1 Detailed Description

Classes and functions for managing event logging.

6.11.2 Enumeration Type Documentation

6.11.2.1 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 28 of file event_manager.h.

6.11.2.2 SystemEvent

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

Represents specific system events.

Enumerator

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

Definition at line 45 of file event_manager.h.

6.11.2.3 PowerEvent

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

Represents specific power-related events.

Enumerator

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

Definition at line 62 of file event_manager.h.

6.11.2.4 CommsEvent

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

Represents specific communication-related events.

Enumerator

RADIO_INIT	Radio initialization event.
RADIO_ERROR	Radio error event.
MSG_RECEIVED	Message received event.
MSG_SENT	Message sent event.
UART_ERROR	UART error event.

Definition at line 85 of file event_manager.h.

6.11.2.5 GPSEvent

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

Represents specific GPS-related events.

Enumerator

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

Definition at line 102 of file event_manager.h.

6.11.2.6 ClockEvent

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

Represents specific clock-related events.

Enumerator

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

Definition at line 126 of file event_manager.h.

6.11.3 Function Documentation

6.11.3.1 __attribute__()

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

6.11.3.2 log_event()

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

Logs an event.

Logs an event to the event buffer.

Parameters

group	The event group.
event	The event identifier.

group	The event group.
event	The event ID.

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

Definition at line 45 of file event_manager.cpp.

Here is the call graph for this function:

6.11.3.3 get_event()

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

Retrieves an event from the event buffer.

Parameters

index	The index of the event to retrieve.

Returns

A const reference to the EventLog at the specified index.

Parameters

index	The index of the event to retrieve.

Returns

A const reference to the EventLog at the specified index. Returns an empty event if the index is out of bounds.

Definition at line 83 of file event_manager.cpp.

6.11.4 Variable Documentation

6.11.4.1 eventLogId

```
volatile uint16_t eventLogId = 0
```

Global event log ID counter.

Definition at line 21 of file event_manager.cpp.

6.11.4.2 systemClock

```
DS3231 systemClock [extern]
```

External declaration of the system clock.

6.11.4.3 eventManager [1/2]

```
EventManagerImpl eventManager
```

Global instance of the EventManager implementation.

Global instance of the EventManagerImpl class.

Definition at line 33 of file event_manager.cpp.

6.11.4.4 __attribute__

```
class EventManager __attribute__
```

6.11.4.5 eventManager [2/2]

```
EventManagerImpl eventManager [extern]
```

Global instance of the EventManagerImpl class.

Global instance of the EventManagerImpl class.

Definition at line 33 of file event_manager.cpp.

6.12 INA3221 Power Monitor

Collaboration diagram for INA3221 Power Monitor:

Topics

- Configuration Functions
- Measurement Functions
- Alert Functions

6.12.1 Detailed Description

6.12.2 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.

6.12.2.1 Detailed Description

Functions for configuring the INA3221 device

6.12.2.2 Function Documentation

6.12.2.2.1 INA3221()

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

Constructor for INA3221 class.

Parameters

addr	I2C address of the device
i2c	Pointer to I2C instance

Definition at line 46 of file INA3221.cpp.

6.12.2.2.2 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:

6.12.2.2.3 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:

6.12.2.2.4 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:

6.12.2.2.5 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:

6.12.2.2.6 read_register()

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

Read a register from the device.

Parameters

reg	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:

6.12.2.2.7 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:

6.12.2.2.8 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:

6.12.2.2.9 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:

6.12.2.2.10 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:

6.12.2.2.11 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:

6.12.2.2.12 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:

6.12.2.2.13 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:

6.12.2.2.14 set_averaging_mode()

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

Set the averaging mode for measurements.

Parameters

mode	Number of samples to average

Definition at line 238 of file INA3221.cpp.

Here is the call graph for this function:

6.12.2.2.15 set_bus_conversion_time()

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

Set bus voltage conversion time.

Parameters

convTime	Conversion time setting

Definition at line 252 of file INA3221.cpp.

Here is the call graph for this function:

6.12.2.2.16 set_shunt_conversion_time()

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

Set shunt voltage conversion time.

Parameters

convTime	Conversion time setting

Definition at line 266 of file INA3221.cpp.

Here is the call graph for this function:

6.12.3 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.

6.12.3.1 Detailed Description

Functions for reading voltage, current and power measurements

6.12.3.2 Function Documentation

6.12.3.2.1 get_shunt_voltage()

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

Get shunt voltage for a specific channel.

Parameters

channel	Channel number (1-3)

Returns

Shunt voltage in microvolts (µV)

Definition at line 282 of file INA3221.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.12.3.2.2 get_current_ma()

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

Get current for a specific channel.

Parameters

channel	Channel number (1-3)

Returns

Current in milliamps (mA)

Definition at line 314 of file INA3221.cpp.

Here is the call graph for this function:

6.12.3.2.3 get_voltage()

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

Get bus voltage for a specific channel.

Parameters

channel	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:

6.12.4 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.

6.12.4.1 Detailed Description

Functions for configuring and reading alert conditions

6.12.4.2 Function Documentation

6.12.4.2.1 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

channel	Channel number (1-3)
voltage↩ _v	Voltage threshold in volts

Definition at line 360 of file INA3221.cpp.

Here is the call graph for this function:

6.12.4.2.2 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

channel	Channel number (1-3)
voltage↩ _v	Voltage threshold in volts

Definition at line 385 of file INA3221.cpp.

Here is the call graph for this function:

6.12.4.2.3 set_power_valid_limit()

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

Set power valid voltage range.

Parameters

voltage_upper↩ _v	Upper voltage threshold in volts
voltage_lower↩ _v	Lower voltage threshold in volts

Definition at line 410 of file INA3221.cpp.

Here is the call graph for this function:

6.12.4.2.4 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:

6.12.4.2.5 get_warn_alert()

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

Get warning alert status for a channel.

Parameters

channel	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:

6.12.4.2.6 get_crit_alert()

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

Get critical alert status for a channel.

Parameters

channel	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:

6.12.4.2.7 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:

6.12.4.2.8 set_alert_latch()

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

Set alert latch mode.

Parameters

enable	true to enable alert latching, false for transparent alerts

Definition at line 497 of file INA3221.cpp.

Here is the call graph for this function:

6.13 Telemetry Manager

Classes

- struct TelemetryRecord

  Structure representing a single telemetry data point.
- struct SensorDataRecord

Functions

- bool telemetry_init ()

  Initialize the telemetry system.
- bool collect_telemetry ()

  Collect telemetry data from sensors and power subsystems.
- bool flush_telemetry ()

  Save buffered telemetry data to storage.
- bool flush_sensor_data ()

  Save buffered sensor data to storage.
- bool is_telemetry_collection_time (uint32_t current_time, uint32_t &last_collection_time)

  Check if it's time to collect telemetry based on interval.
- bool is_telemetry_flush_time (uint32_t &collection_counter)

  Check if it's time to flush telemetry buffer based on count.
- uint32_t get_telemetry_sample_interval ()

  Get the current sample interval in milliseconds.
- void set_telemetry_sample_interval (uint32_t interval_ms)

  Set the telemetry sample interval.
- uint32_t get_telemetry_flush_threshold ()

  Get the number of records before flushing to storage.
- void set_telemetry_flush_threshold (uint32_t records)

  Set the number of records before flushing to storage.
- std::string get_last_telemetry_record_csv ()

  Gets the last telemetry record as a CSV string.
- std::string get_last_sensor_record_csv ()

  Gets the last sensor data record as a CSV string.

Variables

- constexpr int TELEMETRY_BUFFER_SIZE = 20

  Circular buffer for telemetry records.
- TelemetryRecord telemetry_buffer [TELEMETRY_BUFFER_SIZE]

  Circular buffer for telemetry records.
- size_t telemetry_buffer_count
- size_t telemetry_buffer_write_index
- SensorDataRecord sensor_data_buffer [TELEMETRY_BUFFER_SIZE]

  Circular buffer for sensor data records.
- size_t sensor_data_buffer_count
- size_t sensor_data_buffer_write_index
- mutex_t telemetry_mutex

  Mutex for thread-safe access to the telemetry buffer.
- const int TELEMETRY_BUFFER_SIZE

  Circular buffer for telemetry records.

6.13.1 Detailed Description

6.13.2 Function Documentation

6.13.2.1 telemetry_init()

```
bool telemetry_init ()
```

Initialize the telemetry system.

Returns

True if initialization was successful

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

Definition at line 76 of file telemetry_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.2.2 collect_telemetry()

```
bool collect_telemetry ()
```

Collect telemetry data from sensors and power subsystems.

Returns

True if data was successfully collected

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

Definition at line 124 of file telemetry_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.2.3 flush_telemetry()

```
bool flush_telemetry ()
```

Save buffered telemetry data to storage.

Returns

True if data was successfully saved

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

Definition at line 197 of file telemetry_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.2.4 flush_sensor_data()

```
bool flush_sensor_data ()
```

Save buffered sensor data to storage.

Returns

True if data was successfully saved

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

Definition at line 240 of file telemetry_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.2.5 is_telemetry_collection_time()

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

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

Parameters

current_time	Current system time in milliseconds
last_collection_time	Previous collection time in milliseconds

Returns

True if collection interval has passed

Updates last_collection_time if the interval has passed

Definition at line 281 of file telemetry_manager.cpp.

Here is the caller graph for this function:

6.13.2.6 is_telemetry_flush_time()

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

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

Parameters

collection_counter	Current collection counter

Returns

True if flush threshold has been reached

Resets collection_counter to zero if the threshold has been reached

Definition at line 289 of file telemetry_manager.cpp.

Here is the caller graph for this function:

6.13.2.7 get_telemetry_sample_interval()

```
uint32_t get_telemetry_sample_interval ()
```

Get the current sample interval in milliseconds.

Returns

Sample interval in milliseconds

Definition at line 297 of file telemetry_manager.cpp.

6.13.2.8 set_telemetry_sample_interval()

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

Set the telemetry sample interval.

Parameters

interval_ms	New interval in milliseconds

Sets a minimum bound of 100ms to prevent excessive sampling

Definition at line 301 of file telemetry_manager.cpp.

6.13.2.9 get_telemetry_flush_threshold()

```
uint32_t get_telemetry_flush_threshold ()
```

Get the number of records before flushing to storage.

Returns

Number of records in flush threshold

Definition at line 307 of file telemetry_manager.cpp.

6.13.2.10 set_telemetry_flush_threshold()

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

Set the number of records before flushing to storage.

Parameters

records	Number of records in flush threshold

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

Definition at line 311 of file telemetry_manager.cpp.

6.13.2.11 get_last_telemetry_record_csv()

```
std::string get_last_telemetry_record_csv ()
```

Gets the last telemetry record as a CSV string.

Returns

A CSV string representing the last telemetry record, or an empty string if no data is available.

Definition at line 319 of file telemetry_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.2.12 get_last_sensor_record_csv()

```
std::string get_last_sensor_record_csv ()
```

Gets the last sensor data record as a CSV string.

Returns

A CSV string representing the last sensor data record, or an empty string if no data is available.

Definition at line 336 of file telemetry_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.13.3 Variable Documentation

6.13.3.1 TELEMETRY_BUFFER_SIZE [1/2]

```
const int TELEMETRY_BUFFER_SIZE = 20  [constexpr]
```

Circular buffer for telemetry records.

Definition at line 121 of file telemetry_manager.h.

6.13.3.2 telemetry_buffer

```
TelemetryRecord telemetry_buffer[TELEMETRY_BUFFER_SIZE]  [extern]
```

Circular buffer for telemetry records.

Definition at line 60 of file telemetry_manager.cpp.

6.13.3.3 telemetry_buffer_count

```
size_t telemetry_buffer_count  [extern]
```

Definition at line 61 of file telemetry_manager.cpp.

6.13.3.4 telemetry_buffer_write_index

```
size_t telemetry_buffer_write_index [extern]
```

Definition at line 62 of file telemetry_manager.cpp.

6.13.3.5 sensor_data_buffer

```
SensorDataRecord sensor_data_buffer[TELEMETRY_BUFFER_SIZE] [extern]
```

Circular buffer for sensor data records.

Definition at line 67 of file telemetry_manager.cpp.

6.13.3.6 sensor_data_buffer_count

```
size_t sensor_data_buffer_count [extern]
```

Definition at line 68 of file telemetry_manager.cpp.

6.13.3.7 sensor_data_buffer_write_index

```
size_t sensor_data_buffer_write_index [extern]
```

Definition at line 69 of file telemetry_manager.cpp.

6.13.3.8 telemetry_mutex

```
mutex_t telemetry_mutex [extern]
```

Mutex for thread-safe access to the telemetry buffer.

Definition at line 74 of file telemetry_manager.cpp.

6.13.3.9 TELEMETRY_BUFFER_SIZE [2/2]

```
const int TELEMETRY_BUFFER_SIZE [extern], [constexpr]
```

Circular buffer for telemetry records.

Definition at line 122 of file telemetry_manager.h.

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 (SensorDataTypeIdentifier type) override
- bool is_initialized () const override
- SensorType get_type () const override
- bool configure (const std::map< std::string, std::string > &config)
- uint8_t get_address () const override

Public Member Functions inherited from ISensor

- virtual ~ISensor ()=default

Private Attributes

- BH1750 sensor_
- bool initialized_ = false

7.2.1 Detailed Description

Definition at line 9 of file BH1750_WRAPPER.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 BH1750Wrapper()

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

Definition at line 6 of file BH1750_WRAPPER.cpp.

7.2.3 Member Function Documentation

7.2.3.1 get_i2c_addr()

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

7.2.3.2 init()

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

Implements ISensor.

Definition at line 10 of file BH1750_WRAPPER.cpp.

7.2.3.3 read_data()

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

Implements ISensor.

Definition at line 15 of file BH1750_WRAPPER.cpp.

7.2.3.4 is_initialized()

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

Implements ISensor.

Definition at line 22 of file BH1750_WRAPPER.cpp.

7.2.3.5 get_type()

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

Implements ISensor.

Definition at line 26 of file BH1750_WRAPPER.cpp.

7.2.3.6 configure()

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

Implements ISensor.

Definition at line 30 of file BH1750_WRAPPER.cpp.

7.2.3.7 get_address()

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

Implements ISensor.

Definition at line 24 of file BH1750_WRAPPER.h.

7.2.4 Member Data Documentation

7.2.4.1 sensor_

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

Definition at line 11 of file BH1750_WRAPPER.h.

7.2.4.2 initialized_

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

Definition at line 12 of file BH1750_WRAPPER.h.

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

- lib/sensors/BH1750/BH1750_WRAPPER.h
- lib/sensors/BH1750/BH1750_WRAPPER.cpp

7.3 BME280 Class Reference

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

Collaboration diagram for BME280:

Public Member Functions

- BME280 (i2c_inst_t ∗i2cPort, uint8_t address=ADDR_SDO_LOW)
- bool init ()
- void reset ()
- bool read_raw_all (int32_t ∗temperature, int32_t ∗pressure, int32_t ∗humidity)
- float convert_temperature (int32_t temp_raw) const
- float convert_pressure (int32_t pressure_raw) const
- float convert_humidity (int32_t humidity_raw) const

Static Public Attributes

- static constexpr uint8_t ADDR_SDO_LOW = 0x76
- static constexpr uint8_t ADDR_SDO_HIGH = 0x77

Private Member Functions

- bool configure_sensor ()
- bool get_calibration_parameters ()

Private Attributes

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

Static Private Attributes

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

7.3.1 Detailed Description

Definition at line 38 of file BME280.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 BME280()

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

Definition at line 14 of file BME280.cpp.

7.3.3 Member Function Documentation

7.3.3.1 init()

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

Definition at line 18 of file BME280.cpp.

Here is the call graph for this function:

7.3.3.2 reset()

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

Definition at line 59 of file BME280.cpp.

7.3.3.3 read_raw_all()

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

Definition at line 68 of file BME280.cpp.

7.3.3.4 convert_temperature()

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

Definition at line 101 of file BME280.cpp.

7.3.3.5 convert_pressure()

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

Definition at line 110 of file BME280.cpp.

7.3.3.6 convert_humidity()

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

Definition at line 131 of file BME280.cpp.

7.3.3.7 configure_sensor()

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

Definition at line 201 of file BME280.cpp.

Here is the caller graph for this function:

7.3.3.8 get_calibration_parameters()

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

Definition at line 143 of file BME280.cpp.

Here is the caller graph for this function:

7.3.4 Member Data Documentation

7.3.4.1 ADDR_SDO_LOW

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

Definition at line 41 of file BME280.h.

7.3.4.2 ADDR_SDO_HIGH

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

Definition at line 42 of file BME280.h.

7.3.4.3 i2c_port

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

Definition at line 69 of file BME280.h.

7.3.4.4 device_addr

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

Definition at line 70 of file BME280.h.

7.3.4.5 calib_params

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

Definition at line 73 of file BME280.h.

7.3.4.6 initialized_

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

Definition at line 76 of file BME280.h.

7.3.4.7 t_fine

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

Definition at line 79 of file BME280.h.

7.3.4.8 REG_CONFIG

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

Definition at line 82 of file BME280.h.

7.3.4.9 REG_CTRL_MEAS

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

Definition at line 83 of file BME280.h.

7.3.4.10 REG_CTRL_HUM

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

Definition at line 84 of file BME280.h.

7.3.4.11 REG_RESET

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

Definition at line 85 of file BME280.h.

7.3.4.12 REG_PRESSURE_MSB

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

Definition at line 87 of file BME280.h.

7.3.4.13 REG_TEMPERATURE_MSB

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

Definition at line 88 of file BME280.h.

7.3.4.14 REG_HUMIDITY_MSB

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

Definition at line 89 of file BME280.h.

7.3.4.15 REG_DIG_T1_LSB

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

Definition at line 92 of file BME280.h.

7.3.4.16 REG_DIG_T1_MSB

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

Definition at line 93 of file BME280.h.

7.3.4.17 REG_DIG_T2_LSB

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

Definition at line 94 of file BME280.h.

7.3.4.18 REG_DIG_T2_MSB

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

Definition at line 95 of file BME280.h.

7.3.4.19 REG_DIG_T3_LSB

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

Definition at line 96 of file BME280.h.

7.3.4.20 REG_DIG_T3_MSB

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

Definition at line 97 of file BME280.h.

7.3.4.21 REG_DIG_P1_LSB

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

Definition at line 99 of file BME280.h.

7.3.4.22 REG_DIG_P1_MSB

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

Definition at line 100 of file BME280.h.

7.3.4.23 REG_DIG_P2_LSB

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

Definition at line 101 of file BME280.h.

7.3.4.24 REG_DIG_P2_MSB

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

Definition at line 102 of file BME280.h.

7.3.4.25 REG_DIG_P3_LSB

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

Definition at line 103 of file BME280.h.

7.3.4.26 REG_DIG_P3_MSB

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

Definition at line 104 of file BME280.h.

7.3.4.27 REG_DIG_P4_LSB

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

Definition at line 105 of file BME280.h.

7.3.4.28 REG_DIG_P4_MSB

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

Definition at line 106 of file BME280.h.

7.3.4.29 REG_DIG_P5_LSB

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

Definition at line 107 of file BME280.h.

7.3.4.30 REG_DIG_P5_MSB

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

Definition at line 108 of file BME280.h.

7.3.4.31 REG_DIG_P6_LSB

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

Definition at line 109 of file BME280.h.

7.3.4.32 REG_DIG_P6_MSB

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

Definition at line 110 of file BME280.h.

7.3.4.33 REG_DIG_P7_LSB

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

Definition at line 111 of file BME280.h.

7.3.4.34 REG_DIG_P7_MSB

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

Definition at line 112 of file BME280.h.

7.3.4.35 REG_DIG_P8_LSB

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

Definition at line 113 of file BME280.h.

7.3.4.36 REG_DIG_P8_MSB

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

Definition at line 114 of file BME280.h.

7.3.4.37 REG_DIG_P9_LSB

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

Definition at line 115 of file BME280.h.

7.3.4.38 REG_DIG_P9_MSB

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

Definition at line 116 of file BME280.h.

7.3.4.39 REG_DIG_H1

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

Definition at line 119 of file BME280.h.

7.3.4.40 REG_DIG_H2

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

Definition at line 120 of file BME280.h.

7.3.4.41 REG_DIG_H3

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

Definition at line 121 of file BME280.h.

7.3.4.42 REG_DIG_H4

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

Definition at line 122 of file BME280.h.

7.3.4.43 REG_DIG_H5

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

Definition at line 123 of file BME280.h.

7.3.4.44 REG_DIG_H6

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

Definition at line 124 of file BME280.h.

7.3.4.45 NUM_CALIB_PARAMS

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

Definition at line 127 of file BME280.h.

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

- lib/sensors/BME280/BME280.h
- lib/sensors/BME280/BME280.cpp

7.4 BME280CalibParam Struct Reference

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

Public Attributes

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

7.4.1 Detailed Description

Definition at line 11 of file BME280.h.

7.4.2 Member Data Documentation

7.4.2.1 dig_t1

```
uint16_t BME280CalibParam::dig_t1
```

Definition at line 13 of file BME280.h.

7.4.2.2 dig_t2

```
int16_t BME280CalibParam::dig_t2
```

Definition at line 14 of file BME280.h.

7.4.2.3 dig_t3

```
int16_t BME280CalibParam::dig_t3
```

Definition at line 15 of file BME280.h.

7.4.2.4 dig_p1

```
uint16_t BME280CalibParam::dig_p1
```

Definition at line 18 of file BME280.h.

7.4.2.5 dig_p2

```
int16_t BME280CalibParam::dig_p2
```

Definition at line 19 of file BME280.h.

7.4.2.6 dig_p3

```
int16_t BME280CalibParam::dig_p3
```

Definition at line 20 of file BME280.h.

7.4.2.7 dig_p4

```
int16_t BME280CalibParam::dig_p4
```

Definition at line 21 of file BME280.h.

7.4.2.8 dig_p5

```
int16_t BME280CalibParam::dig_p5
```

Definition at line 22 of file BME280.h.

7.4.2.9 dig_p6

```
int16_t BME280CalibParam::dig_p6
```

Definition at line 23 of file BME280.h.

7.4.2.10 dig_p7

```
int16_t BME280CalibParam::dig_p7
```

Definition at line 24 of file BME280.h.

7.4.2.11 dig_p8

```
int16_t BME280CalibParam::dig_p8
```

Definition at line 25 of file BME280.h.

7.4.2.12 dig_p9

```
int16_t BME280CalibParam::dig_p9
```

Definition at line 26 of file BME280.h.

7.4.2.13 dig_h1

```
uint8_t BME280CalibParam::dig_h1
```

Definition at line 29 of file BME280.h.

7.4.2.14 dig_h2

```
int16_t BME280CalibParam::dig_h2
```

Definition at line 30 of file BME280.h.

7.4.2.15 dig_h3

```
uint8_t BME280CalibParam::dig_h3
```

Definition at line 31 of file BME280.h.

7.4.2.16 dig_h4

```
int16_t BME280CalibParam::dig_h4
```

Definition at line 32 of file BME280.h.

7.4.2.17 dig_h5

```
int16_t BME280CalibParam::dig_h5
```

Definition at line 33 of file BME280.h.

7.4.2.18 dig_h6

```
int8_t BME280CalibParam::dig_h6
```

Definition at line 34 of file BME280.h.

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

- lib/sensors/BME280/BME280.h

7.5 BME280Wrapper Class Reference

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

Inheritance diagram for BME280Wrapper:

Collaboration diagram for BME280Wrapper:

Public Member Functions

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

Public Member Functions inherited from ISensor

- virtual ~ISensor ()=default

Private Attributes

- BME280 sensor_
- bool initialized_ = false

7.5.1 Detailed Description

Definition at line 8 of file BME280_WRAPPER.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 BME280Wrapper()

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

Definition at line 3 of file BME280_WRAPPER.cpp.

7.5.3 Member Function Documentation

7.5.3.1 init()

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

Implements ISensor.

Definition at line 5 of file BME280_WRAPPER.cpp.

7.5.3.2 read_data()

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

Implements ISensor.

Definition at line 10 of file BME280_WRAPPER.cpp.

7.5.3.3 is_initialized()

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

Implements ISensor.

Definition at line 26 of file BME280_WRAPPER.cpp.

7.5.3.4 get_type()

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

Implements ISensor.

Definition at line 30 of file BME280_WRAPPER.cpp.

7.5.3.5 configure()

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

Implements ISensor.

Definition at line 34 of file BME280_WRAPPER.cpp.

7.5.3.6 get_address()

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

Implements ISensor.

Definition at line 22 of file BME280_WRAPPER.h.

7.5.4 Member Data Documentation

7.5.4.1 sensor_

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

Definition at line 10 of file BME280_WRAPPER.h.

7.5.4.2 initialized_

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

Definition at line 11 of file BME280_WRAPPER.h.

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

- lib/sensors/BME280/BME280_WRAPPER.h
- lib/sensors/BME280/BME280_WRAPPER.cpp

7.6 INA3221::conf_reg_t Struct Reference

Configuration register bit fields.

Public Attributes

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

7.6.1 Detailed Description

Configuration register bit fields.

Definition at line 101 of file INA3221.h.

7.6.2 Member Data Documentation

7.6.2.1 mode_shunt_en

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

Definition at line 102 of file INA3221.h.

7.6.2.2 mode_bus_en

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

Definition at line 103 of file INA3221.h.

7.6.2.3 mode_continious_en

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

Definition at line 104 of file INA3221.h.

7.6.2.4 shunt_conv_time

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

Definition at line 105 of file INA3221.h.

7.6.2.5 bus_conv_time

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

Definition at line 106 of file INA3221.h.

7.6.2.6 avg_mode

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

Definition at line 107 of file INA3221.h.

7.6.2.7 ch3_en

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

Definition at line 108 of file INA3221.h.

7.6.2.8 ch2_en

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

Definition at line 109 of file INA3221.h.

7.6.2.9 ch1_en

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

Definition at line 110 of file INA3221.h.

7.6.2.10 reset

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

Definition at line 111 of file INA3221.h.

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

- lib/powerman/INA3221/INA3221.h

7.7 DS3231 Class Reference

Class for interfacing with the DS3231 real-time clock.

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

Public Member Functions

- DS3231 (i2c_inst_t *i2c_instance)

  Constructor for the DS3231 class.
- int set_time (ds3231_data_t *data)

  Sets the time on the DS3231 clock.
- int get_time (ds3231_data_t *data)

  Gets the current time from the DS3231 clock.
- int read_temperature (float *resolution)

  Reads the current temperature from the DS3231.
- int set_unix_time (time_t unix_time)

  Sets the time using a Unix timestamp.
- time_t get_unix_time ()

  Gets the current time as a Unix timestamp.
- int clock_enable ()

  Enables the DS3231 clock oscillator.
- int16_t get_timezone_offset () const

  Gets the current timezone offset.
- void set_timezone_offset (int16_t offset_minutes)

  Sets the timezone offset.
- uint32_t get_clock_sync_interval () const

  Gets the clock synchronization interval.
- void set_clock_sync_interval (uint32_t interval_minutes)

  Sets the clock synchronization interval.
- time_t get_last_sync_time () const

  Gets the timestamp of the last clock synchronization.
- void update_last_sync_time ()

  Updates the last sync time to current time.
- time_t get_local_time ()

  Gets the current local time (including timezone offset)
- bool is_sync_needed ()

  Checks if clock synchronization is needed.
- bool sync_clock_with_gps ()

  Synchronizes clock with GPS data.

Private Member Functions

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

  Reads data from a specific register on the DS3231.
- int i2c_write_reg (uint8_t reg_addr, size_t length, uint8_t *data)

  Writes data to a specific register on the DS3231.
- uint8_t bin_to_bcd (const uint8_t data)

  Converts binary value to BCD (Binary Coded Decimal)
- uint8_t bcd_to_bin (const uint8_t bcd)

  Converts BCD (Binary Coded Decimal) to binary value.

Private Attributes

- i2c_inst_t * i2c

  Pointer to I2C instance.
- uint8_t ds3231_addr

  I2C address of DS3231.
- recursive_mutex_t clock_mutex_

  Mutex for thread-safe I2C access.
- int16_t timezone_offset_minutes_ = 60

  Timezone offset in minutes, default: UTC.
- uint32_t sync_interval_minutes_ = 1440

  Sync interval in minutes, default: 24 hours.
- time_t last_sync_time_ = 0

  Last sync timestamp, 0 = never synced.

7.7.1 Detailed Description

Class for interfacing with the DS3231 real-time clock.

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

Definition at line 108 of file DS3231.h.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 DS3231()

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

Constructor for the DS3231 class.

Parameters

in	*i2c_instance*	Pointer to the I2C instance to use
in	*i2c_instance*	Pointer to the i2c_inst_t structure representing the I^2C port

Initializes a DS3231 RTC controller object with the specified I^2C port and default device address. This constructor also initializes the clock mutex for thread-safe I^2C access.

Definition at line 16 of file DS3231.cpp.

7.7.3 Member Function Documentation

7.7.3.1 set_time()

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

Sets the time on the DS3231 clock.

Sets the time on the DS3231 RTC module.

Parameters

in	*data*	Pointer to a ds3231_data_t structure with time information

Returns

0 on success, -1 on failure

Parameters

in	*data*	Pointer to a ds3231_data_t structure containing the time to set

Returns

0 on success, -1 on failure

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

Note

The ds3231_data_t structure must contain valid values for seconds, minutes, hours, day, date, month, year, and century.

Definition at line 33 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.2 get_time()

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

Gets the current time from the DS3231 clock.

Reads the current time from the DS3231 RTC module.

Parameters

out	*data*	Pointer to a ds3231_data_t structure to store time information

Returns

0 on success, -1 on failure

Parameters

out	*data*	Pointer to a ds3231_data_t structure to store the read time

Returns

0 on success, -1 on failure

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

Note

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

Definition at line 102 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.3 read_temperature()

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

Reads the current temperature from the DS3231.

Reads the temperature from the DS3231's internal temperature sensor.

Parameters

out	*resolution*	Pointer to store the temperature value in Celsius

Returns

0 on success, -1 on failure

Parameters

out	*resolution*	Pointer to a float to store the temperature value

Returns

0 on success, -1 on failure

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

Definition at line 143 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.4 set_unix_time()

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

Sets the time using a Unix timestamp.

Sets the DS3231 clock using a Unix timestamp.

Parameters

in	*unix_time*	Time as seconds since Unix epoch (1970-01-01 00:00:00 UTC)

Returns

0 on success, -1 on failure

Parameters

in	*unix_time*	The time in seconds since the Unix epoch (1970-01-01 00:00:00 UTC)

Returns

0 on success, -1 on failure

Converts the provided Unix timestamp to a calendar date and time and sets the DS3231 RTC accordingly. This function properly handles the conversion between the tm structure (used by C standard library) and the internal ds3231_data_t format.

Definition at line 172 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.5 get_unix_time()

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

Gets the current time as a Unix timestamp.

Gets the current time from DS3231 as a Unix timestamp.

Returns

Unix timestamp, or -1 on error

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

Reads the current time from the DS3231 RTC and converts it to a Unix timestamp. This function properly handles the conversion between the internal ds3231_data_t format and the tm structure used by the C standard library.

Definition at line 202 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.6 clock_enable()

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

Enables the DS3231 clock oscillator.

Enables the DS3231's oscillator.

Returns

0 on success, -1 on failure

0 on success, -1 on failure

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

Definition at line 239 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.7 get_timezone_offset()

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

Gets the current timezone offset.

Gets the currently configured timezone offset.

Returns

Timezone offset in minutes (-720 to +720)

The timezone offset in minutes

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

Definition at line 271 of file DS3231.cpp.

7.7.3.8 set_timezone_offset()

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

Sets the timezone offset.

Parameters

	offset_minutes	Offset in minutes (-720 to +720)
in	*offset_minutes*	The timezone offset in minutes

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

Note

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

Definition at line 286 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.9 get_clock_sync_interval()

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

Gets the clock synchronization interval.

Gets the currently configured clock synchronization interval.

Returns

Sync interval in minutes

The sync interval in minutes

Returns the current interval between clock synchronization attempts. This is the time after which is_sync_needed() will return true.

Definition at line 303 of file DS3231.cpp.

7.7.3.10 set_clock_sync_interval()

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

Sets the clock synchronization interval.

Parameters

	interval_minutes	Interval in minutes (1-43200)
in	*interval_minutes*	The desired sync interval in minutes

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

Note

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

Definition at line 317 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.11 get_last_sync_time()

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

Gets the timestamp of the last clock synchronization.

Gets the timestamp of the last successful clock synchronization.

Returns

Unix timestamp of last sync, 0 if never synced

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

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

Definition at line 334 of file DS3231.cpp.

7.7.3.12 update_last_sync_time()

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

Updates the last sync time to current time.

Updates the last sync timestamp to the current time.

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

Definition at line 347 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.13 get_local_time()

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

Gets the current local time (including timezone offset)

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

Returns

Unix timestamp adjusted for timezone, or -1 on error

Local time as Unix timestamp, or -1 on error

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

Definition at line 360 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.14 is_sync_needed()

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

Checks if clock synchronization is needed.

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

Returns

true if sync interval has elapsed since last sync, false otherwise

true if synchronization is needed, false otherwise

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

Definition at line 379 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.15 sync_clock_with_gps()

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

Synchronizes clock with GPS data.

Synchronizes the RTC with time from GPS.

Returns

true if sync successful, false otherwise

Parameters

in	*nmea_data*	Reference to NMEA data containing time information

Returns

true if synchronization succeeded, false if it failed

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

Note

This function emits events to the EventEmitter system that can be monitored by other components of the system.

Definition at line 409 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.16 i2c_read_reg()

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

Reads data from a specific register on the DS3231.

Parameters

in	*reg_addr*	Register address to read from
in	*length*	Number of bytes to read
out	*data*	Buffer to store read data

Returns

0 on success, -1 on failure

Parameters

in	*reg_addr*	Register address to read from
in	*length*	Number of bytes to read
out	*data*	Buffer to store read data

Returns

0 on success, -1 on failure

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

Note

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

Definition at line 455 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.17 i2c_write_reg()

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

Writes data to a specific register on the DS3231.

Parameters

in	*reg_addr*	Register address to write to
in	*length*	Number of bytes to write
in	*data*	Buffer containing data to write

Returns

0 on success, -1 on failure

Parameters

in	*reg_addr*	Register address to write to
in	*length*	Number of bytes to write
in	*data*	Buffer containing data to write

Returns

0 on success, -1 on failure

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

Note

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

Definition at line 496 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.18 bin_to_bcd()

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

Converts binary value to BCD (Binary Coded Decimal)

Converts binary value to Binary-Coded Decimal (BCD) format.

Parameters

in	*data*	Binary value to convert (0-99)

Returns

BCD representation of the input value

Parameters

in	*data*	Binary value to convert (0-99)

Returns

BCD representation of the input value

The DS3231 stores time values in BCD format where each nibble represents a decimal digit. This function converts a standard binary value to its BCD equivalent (e.g., 42 becomes 0x42).

Definition at line 526 of file DS3231.cpp.

Here is the caller graph for this function:

7.7.3.19 bcd_to_bin()

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

Converts BCD (Binary Coded Decimal) to binary value.

Converts Binary-Coded Decimal (BCD) to binary value.

Parameters

in	*bcd*	BCD value to convert

Returns

Binary representation of the input BCD value

Parameters

in	*bcd*	BCD value to convert

Returns

Binary representation of the input BCD value

The DS3231 stores time values in BCD format where each nibble represents a decimal digit. This function converts a BCD value to its standard binary equivalent (e.g., 0x42 becomes 42).

Definition at line 542 of file DS3231.cpp.

Here is the caller graph for this function:

7.7.4 Member Data Documentation

7.7.4.1 i2c

```
i2c_inst_t* DS3231::i2c  [private]
```

Pointer to I2C instance.

Definition at line 226 of file DS3231.h.

7.7.4.2 ds3231_addr

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

I2C address of DS3231.

Definition at line 227 of file DS3231.h.

7.7.4.3 clock_mutex_

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

Mutex for thread-safe I2C access.

Definition at line 265 of file DS3231.h.

7.7.4.4 timezone_offset_minutes_

```
int16_t DS3231::timezone_offset_minutes_ = 60  [private]
```

Timezone offset in minutes, default: UTC.

Definition at line 266 of file DS3231.h.

7.7.4.5 sync_interval_minutes_

```
uint32_t DS3231::sync_interval_minutes_ = 1440  [private]
```

Sync interval in minutes, default: 24 hours.

Definition at line 267 of file DS3231.h.

7.7.4.6 last_sync_time_

```
time_t DS3231::last_sync_time_ = 0  [private]
```

Last sync timestamp, 0 = never synced.

Definition at line 268 of file DS3231.h.

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

- lib/clock/DS3231.h
- lib/clock/DS3231.cpp

7.8 ds3231_data_t Struct Reference

Structure to hold time and date information from DS3231.

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

Public Attributes

- uint8_t seconds

  Seconds (0-59)
- uint8_t minutes

  Minutes (0-59)
- uint8_t hours

  Hours (0-23)
- uint8_t day

  Day of the week (1-7)
- uint8_t date

  Date (1-31)
- uint8_t month

  Month (1-12)
- uint8_t year

  Year (0-99)
- bool century

  Century flag (0-1)

7.8.1 Detailed Description

Structure to hold time and date information from DS3231.

Definition at line 90 of file DS3231.h.

7.8.2 Member Data Documentation

7.8.2.1 seconds

```
uint8_t ds3231_data_t::seconds
```

Seconds (0-59)

Definition at line 91 of file DS3231.h.

7.8.2.2 minutes

```
uint8_t ds3231_data_t::minutes
```

Minutes (0-59)

Definition at line 92 of file DS3231.h.

7.8.2.3 hours

```
uint8_t ds3231_data_t::hours
```

Hours (0-23)

Definition at line 93 of file DS3231.h.

7.8.2.4 day

```
uint8_t ds3231_data_t::day
```

Day of the week (1-7)

Definition at line 94 of file DS3231.h.

7.8.2.5 date

```
uint8_t ds3231_data_t::date
```

Date (1-31)

Definition at line 95 of file DS3231.h.

7.8.2.6 month

```
uint8_t ds3231_data_t::month
```

Month (1-12)

Definition at line 96 of file DS3231.h.

7.8.2.7 year

```
uint8_t ds3231_data_t::year
```

Year (0-99)

Definition at line 97 of file DS3231.h.

7.8.2.8 century

```
bool ds3231_data_t::century
```

Century flag (0-1)

Definition at line 98 of file DS3231.h.

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

- lib/clock/DS3231.h

7.9 EventEmitter Class Reference

Provides a static method for emitting events.

```
#include <event_manager.h>
```

Static Public Member Functions

- template<typename T>
  static void emit (EventGroup group, T event)

  Emits an event.

7.9.1 Detailed Description

Provides a static method for emitting events.

Definition at line 317 of file event_manager.h.

7.9.2 Member Function Documentation

7.9.2.1 emit()

```
template<typename T>
static void EventEmitter::emit (
            EventGroup group,
            T event)  [inline], [static]
```

Emits an event.

Parameters

group	The event group.
event	The event identifier.

Template Parameters

T	The type of the event identifier.

Definition at line 326 of file event_manager.h.

Here is the caller graph for this function:

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

- lib/eventman/event_manager.h

7.10 EventLog Class Reference

Represents a single event log entry.

```
#include <event_manager.h>
```

Public Member Functions

- std::string to_string () const

  Converts the EventLog to a string representation.

Public Attributes

- uint16_t id

  Sequence number.
- uint32_t timestamp

  Unix timestamp or system time.
- uint8_t group

  Event group identifier.
- uint8_t event

  Specific event identifier.

7.10.1 Detailed Description

Represents a single event log entry.

Definition at line 140 of file event_manager.h.

7.10.2 Member Function Documentation

7.10.2.1 to_string()

```
std::string EventLog::to_string () const  [inline]
```

Converts the EventLog to a string representation.

Returns

A string representation of the EventLog.

Definition at line 155 of file event_manager.h.

Here is the caller graph for this function:

7.10.3 Member Data Documentation

7.10.3.1 id

```
uint16_t EventLog::id
```

Sequence number.

Definition at line 143 of file event_manager.h.

7.10.3.2 timestamp

```
uint32_t EventLog::timestamp
```

Unix timestamp or system time.

Definition at line 145 of file event_manager.h.

7.10.3.3 group

```
uint8_t EventLog::group
```

Event group identifier.

Definition at line 147 of file event_manager.h.

7.10.3.4 event

```
uint8_t EventLog::event
```

Specific event identifier.

Definition at line 149 of file event_manager.h.

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

- lib/eventman/event_manager.h

7.11 EventManager Class Reference

Manages the event logging system.

```
#include <event_manager.h>
```

Inheritance diagram for EventManager:

Collaboration diagram for EventManager:

Public Member Functions

- EventManager ()

  Constructor for the EventManager.
- virtual ~EventManager ()=default

  Virtual destructor for the EventManager.
- virtual void init ()

  Initializes the EventManager.
- void log_event (uint8_t group, uint8_t event)

  Logs an event.
- const EventLog & get_event (size_t index) const

  Retrieves an event from the event buffer.
- size_t get_event_count () const

  Gets the number of events in the buffer.
- virtual bool save_to_storage ()=0

  Saves the events to storage.
- virtual bool load_from_storage ()=0

  Loads the events from storage.

Protected Attributes

- EventLog events [EVENT_BUFFER_SIZE]

  Event buffer.
- size_t eventCount

  Number of events in the buffer.
- size_t writeIndex

  Index of the next event to be written.
- mutex_t eventMutex

  Mutex for protecting the event buffer.
- size_t eventsSinceFlush

  Number of events since last flush to storage.

Static Protected Attributes

- static uint16_t nextEventId = 0

  Static event ID counter.

7.11.1 Detailed Description

Manages the event logging system.

Definition at line 169 of file event_manager.h.

7.11.2 Constructor & Destructor Documentation

7.11.2.1 EventManager()

```
EventManager::EventManager () [inline]
```

Constructor for the EventManager.

Initializes the event buffer, mutex, and other internal variables.

Definition at line 175 of file event_manager.h.

7.11.2.2 ~EventManager()

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

Virtual destructor for the EventManager.

7.11.3 Member Function Documentation

7.11.3.1 init()

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

Initializes the EventManager.

Loads events from storage.

Definition at line 192 of file event_manager.h.

Here is the call graph for this function:

Here is the caller graph for this function:

7.11.3.2 get_event_count()

```
size_t EventManager::get_event_count () const [inline]
```

Gets the number of events in the buffer.

Returns

The number of events in the buffer.

Definition at line 214 of file event_manager.h.

7.11.3.3 save_to_storage()

```
virtual bool EventManager::save_to_storage () [pure virtual]
```

Saves the events to storage.

Returns

True if the events were successfully saved, false otherwise.

Implemented in EventManagerImpl.

Here is the caller graph for this function:

7.11.3.4 load_from_storage()

```
virtual bool EventManager::load_from_storage () [pure virtual]
```

Loads the events from storage.

Returns

True if the events were successfully loaded, false otherwise.

Implemented in EventManagerImpl.

Here is the caller graph for this function:

7.11.4 Member Data Documentation

7.11.4.1 events

```
EventLog EventManager::events[EVENT_BUFFER_SIZE] [protected]
```

Event buffer.

Definition at line 230 of file event_manager.h.

7.11.4.2 eventCount

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

Number of events in the buffer.

Definition at line 232 of file event_manager.h.

7.11.4.3 writeIndex

```
size_t EventManager::writeIndex [protected]
```

Index of the next event to be written.

Definition at line 234 of file event_manager.h.

7.11.4.4 eventMutex

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

Mutex for protecting the event buffer.

Definition at line 236 of file event_manager.h.

7.11.4.5 nextEventId

```
uint16_t EventManager::nextEventId = 0 [static], [protected]
```

Static event ID counter.

Definition at line 238 of file event_manager.h.

7.11.4.6 eventsSinceFlush

```
size_t EventManager::eventsSinceFlush [protected]
```

Number of events since last flush to storage.

Definition at line 240 of file event_manager.h.

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

- lib/eventman/event_manager.h
- lib/eventman/event_manager.cpp

7.12 EventManagerImpl Class Reference

Implementation of the EventManager class.

```
#include <event_manager.h>
```

Inheritance diagram for EventManagerImpl:

Collaboration diagram for EventManagerImpl:

Public Member Functions

- EventManagerImpl ()

  Constructor for the EventManagerImpl.
- bool save_to_storage () override

  Saves the events to storage.
- bool load_from_storage () override

  Loads the events from storage.

Public Member Functions inherited from EventManager

- EventManager ()

  Constructor for the EventManager.
- virtual ~EventManager ()=default

  Virtual destructor for the EventManager.
- virtual void init ()

  Initializes the EventManager.
- void log_event (uint8_t group, uint8_t event)

  Logs an event.
- const EventLog & get_event (size_t index) const

  Retrieves an event from the event buffer.
- size_t get_event_count () const

  Gets the number of events in the buffer.

Additional Inherited Members

Protected Attributes inherited from EventManager

- EventLog events [EVENT_BUFFER_SIZE]

  Event buffer.
- size_t eventCount

  Number of events in the buffer.
- size_t writeIndex

  Index of the next event to be written.
- mutex_t eventMutex

  Mutex for protecting the event buffer.
- size_t eventsSinceFlush

  Number of events since last flush to storage.

Static Protected Attributes inherited from EventManager

- static uint16_t nextEventId = 0

  Static event ID counter.

7.12.1 Detailed Description

Implementation of the EventManager class.

Definition at line 248 of file event_manager.h.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 EventManagerImpl()

```
EventManagerImpl::EventManagerImpl () [inline]
```

Constructor for the EventManagerImpl.

Initializes the EventManagerImpl and calls the init method.

Definition at line 254 of file event_manager.h.

Here is the call graph for this function:

7.12.3 Member Function Documentation

7.12.3.1 save_to_storage()

```
bool EventManagerImpl::save_to_storage () [inline], [override], [virtual]
```

Saves the events to storage.

Returns

True if the events were successfully saved, false otherwise.

This method is not yet implemented.

Implements EventManager.

Definition at line 264 of file event_manager.h.

Here is the call graph for this function:

7.12.3.2 load_from_storage()

```
bool EventManagerImpl::load_from_storage () [inline], [override], [virtual]
```

Loads the events from storage.

Returns

True if the events were successfully loaded, false otherwise.

This method is not yet implemented.

Implements EventManager.

Definition at line 301 of file event_manager.h.

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

- lib/eventman/event_manager.h

7.13 FileHandle Struct Reference

```
#include <storage.h>
```

Public Attributes

- int fd
- bool is_open

7.13.1 Detailed Description

Definition at line 19 of file storage.h.

7.13.2 Member Data Documentation

7.13.2.1 fd

```
int FileHandle::fd
```

Definition at line 20 of file storage.h.

7.13.2.2 is_open

```
bool FileHandle::is_open
```

Definition at line 21 of file storage.h.

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

- lib/storage/storage.h

7.14 Frame Struct Reference

Represents a communication frame used for data exchange.

```
#include <protocol.h>
```

Public Attributes

- std::string header
- uint8_t direction
- OperationType operationType
- uint8_t group
- uint8_t command
- std::string value
- std::string unit
- std::string footer

7.14.1 Detailed Description

Represents a communication frame used for data exchange.

This structure encapsulates the different components of a communication frame, including the header, direction, operation type, group ID, command ID, payload value, unit, and footer. It is used for both encoding and decoding messages.

Note

The `header` and `footer` fields are used to mark the beginning and end of the frame, respectively.

The `direction` field indicates the direction of the communication (0 = ground->sat, 1 = sat->ground).

The `operationType` field specifies the type of operation being performed (e.g., GET, SET, ANS, ERR, INF).

The `group` and `command` fields identify the specific command being executed.

The `value` field contains the payload data.

The `unit` field specifies the unit of measurement for the payload data.

Example Usage:

```
// Creating a Frame instance
Frame myFrame;
myFrame.header = FRAME_BEGIN;
myFrame.direction = 1;
myFrame.operationType = OperationType::ANS;
myFrame.group = 2;
myFrame.command = 5;
myFrame.value = "25.5";
myFrame.unit = "VOLT";
myFrame.footer = FRAME_END;

// Encoding the Frame to a string
std::string encodedFrame = frame_encode(myFrame);
```

Example Instances:

```
// Example of a GET command
Frame getCommand;
getCommand.header = FRAME_BEGIN;
getCommand.direction = 0;
getCommand.operationType = OperationType::GET;
getCommand.group = 1;
getCommand.command = 10;
```

```
getCommand.value = "";
getCommand.unit = "";
getCommand.footer = FRAME_END;

// Example of an ANSWER command
Frame answerCommand;
answerCommand.header = FRAME_BEGIN;
answerCommand.direction = 1;
answerCommand.operationType = OperationType::ANS;
answerCommand.group = 1;
answerCommand.command = 10;
answerCommand.value = "OK";
answerCommand.unit = "";
answerCommand.footer = FRAME_END;
```

Example of Encoded Frames:

```
// Encoded GET command example:
// KBST;0;GET;1;10;;TSBK

// Encoded SET command example:
// KBST;0;SET;2;5;25.5;VOLT;TSBK

// Encoded ANSWER command example:
// KBST;1;ANS;1;10;OK;;TSBK

// Encoded ERROR command example:
// KBST;1;ERR;3;1;Invalid Parameter;;TSBK

// Encoded INFO command example:
// KBST;1;INF;4;2;System Booted;;TSBK
```

Definition at line 227 of file protocol.h.

7.14.2 Member Data Documentation

7.14.2.1 header

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

Definition at line 228 of file protocol.h.

7.14.2.2 direction

```
uint8_t Frame::direction
```

Definition at line 229 of file protocol.h.

7.14.2.3 operationType

```
OperationType Frame::operationType
```

Definition at line 230 of file protocol.h.

7.14.2.4 group

```
uint8_t Frame::group
```

Definition at line 231 of file protocol.h.

7.14.2.5 command

```
uint8_t Frame::command
```

Definition at line 232 of file protocol.h.

7.14.2.6 value

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

Definition at line 233 of file protocol.h.

7.14.2.7 unit

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

Definition at line 234 of file protocol.h.

7.14.2.8 footer

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

Definition at line 235 of file protocol.h.

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

- lib/comms/protocol.h

7.15 HMC5883L Class Reference

```
#include <HMC5883L.h>
```

Public Member Functions

- HMC5883L (i2c_inst_t *i2c, uint8_t address=0x0D)
- bool init ()
- bool read (int16_t &x, int16_t &y, int16_t &z)

Private Member Functions

- bool write_register (uint8_t reg, uint8_t value)
- bool read_register (uint8_t reg, uint8_t *buffer, size_t length)

Private Attributes

- i2c_inst_t * i2c
- uint8_t address

7.15.1 Detailed Description

Definition at line 6 of file HMC5883L.h.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 HMC5883L()

```
HMC5883L::HMC5883L (
            i2c_inst_t * i2c,
            uint8_t address = 0x0D)
```

Definition at line 3 of file HMC5883L.cpp.

7.15.3 Member Function Documentation

7.15.3.1 init()

```
bool HMC5883L::init ()
```

Definition at line 5 of file HMC5883L.cpp.

Here is the call graph for this function:

7.15.3.2 read()

```
bool HMC5883L::read (
            int16_t & x,
            int16_t & y,
            int16_t & z)
```

Definition at line 13 of file HMC5883L.cpp.

Here is the call graph for this function:

7.15.3.3 write_register()

```
bool HMC5883L::write_register (
            uint8_t reg,
            uint8_t value)  [private]
```

Definition at line 28 of file HMC5883L.cpp.

Here is the caller graph for this function:

7.15.3.4 read_register()

```
bool HMC5883L::read_register (
            uint8_t reg,
            uint8_t * buffer,
            size_t length)  [private]
```

Definition at line 33 of file HMC5883L.cpp.

Here is the caller graph for this function:

7.15.4 Member Data Documentation

7.15.4.1 i2c

```
i2c_inst_t* HMC5883L::i2c  [private]
```

Definition at line 13 of file HMC5883L.h.

7.15.4.2 address

```
uint8_t HMC5883L::address  [private]
```

Definition at line 14 of file HMC5883L.h.

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

- lib/sensors/HMC5883L/HMC5883L.h
- lib/sensors/HMC5883L/HMC5883L.cpp

7.16 HMC5883LWrapper Class Reference

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

Inheritance diagram for HMC5883LWrapper:

Collaboration diagram for HMC5883LWrapper:

Public Member Functions

- HMC5883LWrapper (i2c_inst_t *i2c)
- bool init () override
- float read_data (SensorDataTypeIdentifier type) override
- bool is_initialized () const override
- SensorType get_type () const override
- bool configure (const std::map< std::string, std::string > &config) override
- uint8_t get_address () const override

Public Member Functions inherited from ISensor

- virtual ~ISensor ()=default

Private Attributes

- HMC5883L sensor_
- bool initialized_

7.16.1 Detailed Description

Definition at line 7 of file HMC5883L_WRAPPER.h.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 HMC5883LWrapper()

```
HMC5883LWrapper::HMC5883LWrapper (
            i2c_inst_t * i2c)
```

Definition at line 5 of file HMC5883L_WRAPPER.cpp.

7.16.3 Member Function Documentation

7.16.3.1 init()

```
bool HMC5883LWrapper::init () [override], [virtual]
```

Implements ISensor.

Definition at line 7 of file HMC5883L_WRAPPER.cpp.

7.16.3.2 read_data()

```
float HMC5883LWrapper::read_data (
            SensorDataTypeIdentifier type) [override], [virtual]
```

Implements ISensor.

Definition at line 12 of file HMC5883L_WRAPPER.cpp.

7.16.3.3 is_initialized()

```
bool HMC5883LWrapper::is_initialized () const [override], [virtual]
```

Implements ISensor.

Definition at line 35 of file HMC5883L_WRAPPER.cpp.

7.16.3.4 get_type()

```
SensorType HMC5883LWrapper::get_type () const [override], [virtual]
```

Implements ISensor.

Definition at line 39 of file HMC5883L_WRAPPER.cpp.

7.16.3.5 configure()

```
bool HMC5883LWrapper::configure (
            const std::map< std::string, std::string > & config) [override], [virtual]
```

Implements ISensor.

Definition at line 43 of file HMC5883L_WRAPPER.cpp.

7.16.3.6 get_address()

```
uint8_t HMC5883LWrapper::get_address () const [inline], [override], [virtual]
```

Implements ISensor.

Definition at line 16 of file HMC5883L_WRAPPER.h.

7.16.4 Member Data Documentation

7.16.4.1 sensor_

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

Definition at line 21 of file HMC5883L_WRAPPER.h.

7.16.4.2 initialized_

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

Definition at line 22 of file HMC5883L_WRAPPER.h.

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

- lib/sensors/HMC5883L/HMC5883L_WRAPPER.h
- lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp

7.17 INA3221 Class Reference

INA3221 Triple-Channel Power Monitor driver class.

```
#include <INA3221.h>
```

Collaboration diagram for INA3221:

Classes

- struct conf_reg_t

  Configuration register bit fields.
- struct masken_reg_t

  Mask/Enable register bit fields.

Public Member Functions

- INA3221 (ina3221_addr_t addr, i2c_inst_t *i2c)

  Constructor for INA3221 class.
- bool begin ()

  Initialize the INA3221 device.
- uint16_t read_register (ina3221_reg_t reg)

  Read a register from the device.
- void reset ()

  Reset the INA3221 to default settings.
- void set_mode_power_down ()

  Set device to power-down mode.
- void set_mode_continuous ()

  Set device to continuous measurement mode.
- void set_mode_triggered ()

  Set device to triggered measurement mode.
- void set_shunt_measurement_enable ()

  Enable shunt voltage measurements.
- void set_shunt_measurement_disable ()

  Disable shunt voltage measurements.
- void set_bus_measurement_enable ()

  Enable bus voltage measurements.
- void set_bus_measurement_disable ()

  Disable bus voltage measurements.
- void set_averaging_mode (ina3221_avg_mode_t mode)

  Set the averaging mode for measurements.
- void set_bus_conversion_time (ina3221_conv_time_t convTime)

  Set bus voltage conversion time.
- void set_shunt_conversion_time (ina3221_conv_time_t convTime)

  Set shunt voltage conversion time.
- uint16_t get_manufacturer_id ()

  Get the manufacturer ID of the device.
- uint16_t get_die_id ()

  Get the die ID of the device.
- int32_t get_shunt_voltage (ina3221_ch_t channel)

  Get shunt voltage for a specific channel.
- float get_current (ina3221_ch_t channel)
- float get_current_ma (ina3221_ch_t channel)

  Get current for a specific channel.
- float get_voltage (ina3221_ch_t channel)

  Get bus voltage for a specific channel.
- void set_warn_alert_limit (ina3221_ch_t channel, float voltage_v)

  Set warning alert voltage threshold for a channel.
- void set_crit_alert_limit (ina3221_ch_t channel, float voltage_v)

  Set critical alert voltage threshold for a channel.
- void set_power_valid_limit (float voltage_upper_v, float voltage_lower_v)

  Set power valid voltage range.
- void enable_alerts ()

  Enable all alert functions.
- bool get_warn_alert (ina3221_ch_t channel)

  Get warning alert status for a channel.

- bool get_crit_alert (ina3221_ch_t channel)

  Get critical alert status for a channel.
- bool get_power_valid_alert ()

  Get power valid alert status.
- void set_alert_latch (bool enable)

  Set alert latch mode.

Private Member Functions

- void _read (ina3221_reg_t reg, uint16_t *val)

  Read a 16-bit register from the device.
- void _write (ina3221_reg_t reg, uint16_t *val)

  Write a 16-bit value to a register.

Private Attributes

- i2c_inst_t * _i2c
- ina3221_addr_t _i2c_addr
- uint32_t _shuntRes [INA3221_CH_NUM]
- uint32_t _filterRes [INA3221_CH_NUM]
- masken_reg_t _masken_reg

7.17.1 Detailed Description

INA3221 Triple-Channel Power Monitor driver class.

Provides functionality for voltage, current, and power monitoring with configurable alerts and power valid monitoring

Definition at line 96 of file INA3221.h.

7.17.2 Member Function Documentation

7.17.2.1 _read()

```
void INA3221::_read (
            ina3221_reg_t reg,
            uint16_t * val)  [private]
```

Read a 16-bit register from the device.

Parameters

reg	Register address
val	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

reg	Register address
val	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
- virtual uint8_t get_address () const =0

7.18.1 Detailed Description

Definition at line 34 of file ISensor.h.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 ~ISensor()

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

7.18.3 Member Function Documentation

7.18.3.1 init()

```
virtual bool ISensor::init () [pure virtual]
```

Implemented in BH1750Wrapper, BME280Wrapper, HMC5883LWrapper, and MPU6050Wrapper.

7.18.3.2 read_data()

```
virtual float ISensor::read_data (
            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.

7.18.3.6 get_address()

```
virtual uint8_t ISensor::get_address () const  [pure virtual]
```

Implemented in BH1750Wrapper, BME280Wrapper, and HMC5883LWrapper.

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

- lib/sensors/ISensor.h

7.19 INA3221::masken_reg_t Struct Reference

Mask/Enable register bit fields.

Public Attributes

- uint16_t conv_ready:1
- uint16_t timing_ctrl_alert:1
- uint16_t pwr_valid_alert:1
- uint16_t warn_alert_ch3:1
- uint16_t warn_alert_ch2:1
- uint16_t warn_alert_ch1:1
- uint16_t shunt_sum_alert:1
- uint16_t crit_alert_ch3:1
- uint16_t crit_alert_ch2:1
- uint16_t crit_alert_ch1:1
- uint16_t crit_alert_latch_en:1
- uint16_t warn_alert_latch_en:1
- uint16_t shunt_sum_en_ch3:1
- uint16_t shunt_sum_en_ch2:1
- uint16_t shunt_sum_en_ch1:1
- uint16_t reserved:1

7.19.1 Detailed Description

Mask/Enable register bit fields.

Definition at line 117 of file INA3221.h.

7.19.2 Member Data Documentation

7.19.2.1 conv_ready

```
uint16_t INA3221::masken_reg_t::conv_ready
```

Definition at line 118 of file INA3221.h.

7.19.2.2 timing_ctrl_alert

```
uint16_t INA3221::masken_reg_t::timing_ctrl_alert
```

Definition at line 119 of file INA3221.h.

7.19.2.3 pwr_valid_alert

```
uint16_t INA3221::masken_reg_t::pwr_valid_alert
```

Definition at line 120 of file INA3221.h.

7.19.2.4 warn_alert_ch3

```
uint16_t INA3221::masken_reg_t::warn_alert_ch3
```

Definition at line 121 of file INA3221.h.

7.19.2.5 warn_alert_ch2

```
uint16_t INA3221::masken_reg_t::warn_alert_ch2
```

Definition at line 122 of file INA3221.h.

7.19.2.6 warn_alert_ch1

```
uint16_t INA3221::masken_reg_t::warn_alert_ch1
```

Definition at line 123 of file INA3221.h.

7.19.2.7 shunt_sum_alert

```
uint16_t INA3221::masken_reg_t::shunt_sum_alert
```

Definition at line 124 of file INA3221.h.

7.19.2.8 crit_alert_ch3

```
uint16_t INA3221::masken_reg_t::crit_alert_ch3
```

Definition at line 125 of file INA3221.h.

7.19.2.9 crit_alert_ch2

```
uint16_t INA3221::masken_reg_t::crit_alert_ch2
```

Definition at line 126 of file INA3221.h.

7.19.2.10 crit_alert_ch1

```
uint16_t INA3221::masken_reg_t::crit_alert_ch1
```

Definition at line 127 of file INA3221.h.

7.19.2.11 crit_alert_latch_en

```
uint16_t INA3221::masken_reg_t::crit_alert_latch_en
```

Definition at line 128 of file INA3221.h.

7.19.2.12 warn_alert_latch_en

```
uint16_t INA3221::masken_reg_t::warn_alert_latch_en
```

Definition at line 129 of file INA3221.h.

7.19.2.13 shunt_sum_en_ch3

```
uint16_t INA3221::masken_reg_t::shunt_sum_en_ch3
```

Definition at line 130 of file INA3221.h.

7.19.2.14 shunt_sum_en_ch2

```
uint16_t INA3221::masken_reg_t::shunt_sum_en_ch2
```

Definition at line 131 of file INA3221.h.

7.19.2.15 shunt_sum_en_ch1

```
uint16_t INA3221::masken_reg_t::shunt_sum_en_ch1
```

Definition at line 132 of file INA3221.h.

7.19.2.16 reserved

```
uint16_t INA3221::masken_reg_t::reserved
```

Definition at line 133 of file INA3221.h.

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

- lib/powerman/INA3221/INA3221.h

7.20 MPU6050Wrapper Class Reference

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

Inheritance diagram for MPU6050Wrapper:

Collaboration diagram for MPU6050Wrapper:

Public Member Functions

- MPU6050Wrapper ()
- bool init () override
- float read_data (SensorDataTypeIdentifier type) override
- bool is_initialized () const override
- SensorType get_type () const override
- bool configure (const std::map< std::string, std::string > &config)

Public Member Functions inherited from ISensor

- virtual ~ISensor ()=default
- virtual uint8_t get_address () const =0

Private Attributes

- MPU6050 sensor_
- bool initialized_ = false

7.20.1 Detailed Description

Definition at line 9 of file MPU6050_WRAPPER.h.

7.20.2 Constructor & Destructor Documentation

7.20.2.1 MPU6050Wrapper()

```
MPU6050Wrapper::MPU6050Wrapper ()
```

7.20.3 Member Function Documentation

7.20.3.1 init()

```
bool MPU6050Wrapper::init ()  [override], [virtual]
```

Implements ISensor.

7.20.3.2 read_data()

```
float MPU6050Wrapper::read_data (
            SensorDataTypeIdentifier type)  [override], [virtual]
```

Implements ISensor.

7.20.3.3 is_initialized()

```
bool MPU6050Wrapper::is_initialized () const  [override], [virtual]
```

Implements ISensor.

7.20.3.4 get_type()

```
SensorType MPU6050Wrapper::get_type () const  [override], [virtual]
```

Implements ISensor.

7.20.3.5 configure()

```
bool MPU6050Wrapper::configure (
            const std::map< std::string, std::string > & config)  [virtual]
```

Implements ISensor.

7.20.4 Member Data Documentation

7.20.4.1 sensor_

```
MPU6050 MPU6050Wrapper::sensor_  [private]
```

Definition at line 11 of file MPU6050_WRAPPER.h.

7.20.4.2 initialized_

```
bool MPU6050Wrapper::initialized_ = false  [private]
```

Definition at line 12 of file MPU6050_WRAPPER.h.

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

- lib/sensors/MPU6050/MPU6050_WRAPPER.h

7.21 NMEAData Class Reference

```
#include <NMEA_data.h>
```

Public Member Functions

- NMEAData ()
- void update_rmc_tokens (const std::vector< std::string > &tokens)
- void update_gga_tokens (const std::vector< std::string > &tokens)
- std::vector< std::string > get_rmc_tokens () const
- std::vector< std::string > get_gga_tokens () const
- bool has_valid_time () const
- time_t get_unix_time () const

Private Attributes

- std::vector< std::string > rmc_tokens_
- std::vector< std::string > gga_tokens_
- mutex_t rmc_mutex_
- mutex_t gga_mutex_

7.21.1 Detailed Description

Definition at line 10 of file NMEA_data.h.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 NMEAData()

```
NMEAData::NMEAData ()
```

Definition at line 5 of file NMEA_data.cpp.

7.21.3 Member Function Documentation

7.21.3.1 update_rmc_tokens()

```
void NMEAData::update_rmc_tokens (
            const std::vector< std::string > & tokens)
```

Definition at line 10 of file NMEA_data.cpp.

7.21.3.2 update_gga_tokens()

```
void NMEAData::update_gga_tokens (
            const std::vector< std::string > & tokens)
```

Definition at line 16 of file NMEA_data.cpp.

7.21.3.3 get_rmc_tokens()

```
std::vector< std::string > NMEAData::get_rmc_tokens () const
```

Definition at line 22 of file NMEA_data.cpp.

7.21.3.4 get_gga_tokens()

```
std::vector< std::string > NMEAData::get_gga_tokens () const
```

Definition at line 29 of file NMEA_data.cpp.

7.21.3.5 has_valid_time()

```
bool NMEAData::has_valid_time () const
```

Definition at line 36 of file NMEA_data.cpp.

Here is the caller graph for this function:

7.21.3.6 get_unix_time()

```
time_t NMEAData::get_unix_time () const
```

Definition at line 40 of file NMEA_data.cpp.

Here is the call graph for this function:

7.21.4 Member Data Documentation

7.21.4.1 rmc_tokens_

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

Definition at line 24 of file NMEA_data.h.

7.21.4.2 gga_tokens_

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

Definition at line 25 of file NMEA_data.h.

7.21.4.3 rmc_mutex_

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

Definition at line 26 of file NMEA_data.h.

7.21.4.4 gga_mutex_

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

Definition at line 27 of file NMEA_data.h.

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

- lib/location/NMEA/NMEA_data.h
- lib/location/NMEA/NMEA_data.cpp

7.22 PowerManager Class Reference

```
#include <PowerManager.h>
```

Collaboration diagram for PowerManager:

Public Member Functions

- PowerManager (i2c_inst_t *i2c)
- bool initialize ()
- std::string read_device_ids ()
- float get_current_charge_solar ()
- float get_current_charge_usb ()
- float get_current_charge_total ()
- float get_current_draw ()
- float get_voltage_battery ()
- float get_voltage_5v ()
- void configure (const std::map< std::string, std::string > &config)
- bool is_charging_solar ()
- bool is_charging_usb ()
- bool check_power_alerts ()

Static Public Attributes

- static constexpr float SOLAR_CURRENT_THRESHOLD = 50.0f
- static constexpr float USB_CURRENT_THRESHOLD = 50.0f
- static constexpr float VOLTAGE_LOW_THRESHOLD = 4.6f
- static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f
- static constexpr float FALL_RATE_THRESHOLD = -0.02f
- static constexpr int FALLING_TREND_REQUIRED = 3

Private Attributes

- INA3221 ina3221_
- bool initialized_
- recursive_mutex_t powerman_mutex_
- bool charging_solar_active_ = false
- bool charging_usb_active_ = false

7.22.1 Detailed Description

Definition at line 11 of file PowerManager.h.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 PowerManager()

```
PowerManager::PowerManager (
            i2c_inst_t * i2c)
```

Definition at line 6 of file PowerManager.cpp.

7.22.3 Member Function Documentation

7.22.3.1 initialize()

```
bool PowerManager::initialize ()
```

Definition at line 11 of file PowerManager.cpp.

7.22.3.2 read_device_ids()

```
std::string PowerManager::read_device_ids ()
```

Definition at line 28 of file PowerManager.cpp.

7.22.3.3 get_current_charge_solar()

```
float PowerManager::get_current_charge_solar ()
```

Definition at line 74 of file PowerManager.cpp.

Here is the caller graph for this function:

7.22.3.4 get_current_charge_usb()

```
float PowerManager::get_current_charge_usb ()
```

Definition at line 58 of file PowerManager.cpp.

Here is the caller graph for this function:

7.22.3.5 get_current_charge_total()

```
float PowerManager::get_current_charge_total ()
```

Definition at line 82 of file PowerManager.cpp.

7.22.3.6 get_current_draw()

```
float PowerManager::get_current_draw ()
```

Definition at line 66 of file PowerManager.cpp.

7.22.3.7 get_voltage_battery()

```
float PowerManager::get_voltage_battery ()
```

Definition at line 42 of file PowerManager.cpp.

7.22.3.8 get_voltage_5v()

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

Definition at line 50 of file PowerManager.cpp.

7.22.3.9 configure()

```
void PowerManager::configure (
            const std::map< std::string, std::string > & config)
```

Definition at line 90 of file PowerManager.cpp.

7.22.3.10 is_charging_solar()

```
bool PowerManager::is_charging_solar ()
```

Definition at line 119 of file PowerManager.cpp.

Here is the call graph for this function:

7.22.3.11 is_charging_usb()

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

Definition at line 127 of file PowerManager.cpp.

Here is the call graph for this function:

7.22.3.12 check_power_alerts()

```
bool PowerManager::check_power_alerts ()
```

Definition at line 135 of file PowerManager.cpp.

Here is the call graph for this function:

7.22.4 Member Data Documentation

7.22.4.1 SOLAR_CURRENT_THRESHOLD

```
float PowerManager::SOLAR_CURRENT_THRESHOLD = 50.0f  [static], [constexpr]
```

Definition at line 28 of file PowerManager.h.

7.22.4.2 USB_CURRENT_THRESHOLD

```
float PowerManager::USB_CURRENT_THRESHOLD = 50.0f  [static], [constexpr]
```

Definition at line 29 of file PowerManager.h.

7.22.4.3 VOLTAGE_LOW_THRESHOLD

```
float PowerManager::VOLTAGE_LOW_THRESHOLD = 4.6f  [static], [constexpr]
```

Definition at line 30 of file PowerManager.h.

7.22.4.4 VOLTAGE_OVERCHARGE_THRESHOLD

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

Definition at line 31 of file PowerManager.h.

7.22.4.5 FALL_RATE_THRESHOLD

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

Definition at line 32 of file PowerManager.h.

7.22.4.6 FALLING_TREND_REQUIRED

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

Definition at line 33 of file PowerManager.h.

7.22.4.7 ina3221_

```
INA3221 PowerManager::ina3221_  [private]
```

Definition at line 36 of file PowerManager.h.

7.22.4.8 initialized_

```
bool PowerManager::initialized_  [private]
```

Definition at line 37 of file PowerManager.h.

7.22.4.9 powerman_mutex_

```
recursive_mutex_t PowerManager::powerman_mutex_  [private]
```

Definition at line 38 of file PowerManager.h.

7.22.4.10 charging_solar_active_

```
bool PowerManager::charging_solar_active_ = false  [private]
```

Definition at line 39 of file PowerManager.h.

7.22.4.11 charging_usb_active_

```
bool PowerManager::charging_usb_active_ = false  [private]
```

Definition at line 40 of file PowerManager.h.

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

- lib/powerman/PowerManager.h
- lib/powerman/PowerManager.cpp

7.23 SensorDataRecord Struct Reference

```
#include <telemetry_manager.h>
```

Public Member Functions

- std::string to_csv () const

Public Attributes

- uint32_t timestamp
- float temperature
- float pressure
- float humidity
- float light

7.23.1 Detailed Description

Definition at line 98 of file telemetry_manager.h.

7.23.2 Member Function Documentation

7.23.2.1 to_csv()

```
std::string SensorDataRecord::to_csv () const  [inline]
```

Definition at line 106 of file telemetry_manager.h.

Here is the caller graph for this function:

7.23.3 Member Data Documentation

7.23.3.1 timestamp

```
uint32_t SensorDataRecord::timestamp
```

Unix timestamp of the record

Definition at line 99 of file telemetry_manager.h.

7.23.3.2 temperature

```
float SensorDataRecord::temperature
```

Temperature in degrees Celsius

Definition at line 100 of file telemetry_manager.h.

7.23.3.3 pressure

```
float SensorDataRecord::pressure
```

Pressure in hPa

Definition at line 101 of file telemetry_manager.h.

7.23.3.4 humidity

```
float SensorDataRecord::humidity
```

Relative humidity in %

Definition at line 102 of file telemetry_manager.h.

7.23.3.5 light

```
float SensorDataRecord::light
```

Light intensity in lux

Definition at line 103 of file telemetry_manager.h.

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

- lib/telemetry/telemetry_manager.h

7.24 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, SensorDataTypeIdentifier dataType)

  Reads a specific data type (e.g., temperature, humidity) from a sensor.
- ISensor * get_sensor (SensorType type)
- std::vector< std::pair< SensorType, uint8_t > > scan_connected_sensors (i2c_inst_t *i2c)

  Scans the I2C bus for connected sensors.
- std::vector< std::pair< SensorType, uint8_t > > get_available_sensors ()

  Retrieves a list of available sensor types with their addresses.

Static Public Member Functions

- static SensorWrapper & get_instance ()

  Provides a global instance of SensorWrapper.

Private Member Functions

- SensorWrapper ()

  Default constructor for SensorWrapper.

Private Attributes

- std::map< SensorType, ISensor * > sensors

7.24.1 Detailed Description

Manages different sensor types and provides a unified interface for accessing sensor data.

Definition at line 45 of file ISensor.h.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 SensorWrapper()

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

Default constructor for SensorWrapper.

Here is the caller graph for this function:

7.24.3 Member Function Documentation

7.24.3.1 get_instance()

```
SensorWrapper & SensorWrapper::get_instance () [static]
```

Provides a global instance of SensorWrapper.

Returns

A reference to the single SensorWrapper instance.

Definition at line 24 of file ISensor.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.3.2 sensor_init()

```
bool SensorWrapper::sensor_init (
            SensorType type,
            i2c_inst_t * i2c = nullptr)
```

Initializes a given sensor type on the specified I2C bus.

Parameters

type	The sensor type (LIGHT, ENVIRONMENT, etc.).
i2c	The I2C interface pointer.

Returns

True if initialization succeeded, otherwise false.

Definition at line 42 of file ISensor.cpp.

7.24.3.3 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

type	The sensor type.
config	Key-value pairs for sensor configuration.

Returns

True if the sensor was successfully configured, otherwise false.

Definition at line 67 of file ISensor.cpp.

Here is the caller graph for this function:

7.24.3.4 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

sensorType	The sensor type.
dataType	The type of data to read (light level, temperature, etc.).

Returns

The requested measurement. Returns 0.0f if sensor not found or uninitialized.

Definition at line 83 of file ISensor.cpp.

Here is the caller graph for this function:

7.24.3.5 get_sensor()

```
ISensor * SensorWrapper::get_sensor (
            SensorType type)
```

7.24.3.6 scan_connected_sensors()

```
std::vector< std::pair< SensorType, uint8_t > > SensorWrapper::scan_connected_sensors (
            i2c_inst_t * i2c)
```

Scans the I2C bus for connected sensors.

Parameters

i2c	The I2C interface to scan

Returns

Vector of pairs containing sensor type and I2C address of detected sensors

Definition at line 114 of file ISensor.cpp.

Here is the call graph for this function:

7.24.3.7 get_available_sensors()

```
std::vector< std::pair< SensorType, uint8_t > > SensorWrapper::get_available_sensors ()
```

Retrieves a list of available sensor types with their addresses.

Returns

A vector of pairs containing sensor type and I2C address

Definition at line 96 of file ISensor.cpp.

Here is the caller graph for this function:

7.24.4 Member Data Documentation

7.24.4.1 sensors

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

Definition at line 57 of file ISensor.h.

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

- lib/sensors/ISensor.h
- lib/sensors/ISensor.cpp

7.25 TelemetryRecord Struct Reference

Structure representing a single telemetry data point.

```
#include <telemetry_manager.h>
```

Public Member Functions

- std::string to_csv () const

Public Attributes

- uint32_t timestamp
- std::string build_version
- float battery_voltage
- float system_voltage
- float charge_current_usb
- float charge_current_solar
- float discharge_current
- std::string time
- std::string latitude
- std::string lat_dir
- std::string longitude
- std::string lon_dir
- std::string speed
- std::string course
- std::string date
- std::string fix_quality
- std::string satellites
- std::string altitude

7.25.1 Detailed Description

Structure representing a single telemetry data point.

Contains all measurements from power subsystem, sensors, and GPS data collected at a specific point in time

Definition at line 42 of file telemetry_manager.h.

7.25.2 Member Function Documentation

7.25.2.1 to_csv()

```
std::string TelemetryRecord::to_csv () const  [inline]
```

Definition at line 71 of file telemetry_manager.h.

Here is the caller graph for this function:

7.25.3 Member Data Documentation

7.25.3.1 timestamp

```
uint32_t TelemetryRecord::timestamp
```

Unix timestamp of the record

Definition at line 43 of file telemetry_manager.h.

7.25.3.2 build_version

```
std::string TelemetryRecord::build_version
```

Build version of the firmware

Definition at line 45 of file telemetry_manager.h.

7.25.3.3 battery_voltage

```
float TelemetryRecord::battery_voltage
```

Battery voltage in volts

Definition at line 48 of file telemetry_manager.h.

7.25.3.4 system_voltage

```
float TelemetryRecord::system_voltage
```

System 5V rail voltage in volts

Definition at line 49 of file telemetry_manager.h.

7.25.3.5 charge_current_usb

```
float TelemetryRecord::charge_current_usb
```

USB charging current in mA

Definition at line 50 of file telemetry_manager.h.

7.25.3.6 charge_current_solar

```
float TelemetryRecord::charge_current_solar
```

Solar charging current in mA

Definition at line 51 of file telemetry_manager.h.

7.25.3.7 discharge_current

```
float TelemetryRecord::discharge_current
```

Battery discharge current in mA

Definition at line 52 of file telemetry_manager.h.

7.25.3.8 time

```
std::string TelemetryRecord::time
```

UTC time from GPS

Definition at line 55 of file telemetry_manager.h.

7.25.3.9 latitude

```
std::string TelemetryRecord::latitude
```

Latitude from GPS

Definition at line 56 of file telemetry_manager.h.

7.25.3.10 lat_dir

```
std::string TelemetryRecord::lat_dir
```

N/S latitude direction

Definition at line 57 of file telemetry_manager.h.

7.25.3.11 longitude

```
std::string TelemetryRecord::longitude
```

Longitude from GPS

Definition at line 58 of file telemetry_manager.h.

7.25.3.12 lon_dir

```
std::string TelemetryRecord::lon_dir
```

E/W longitude direction

Definition at line 59 of file telemetry_manager.h.

7.25.3.13 speed

```
std::string TelemetryRecord::speed
```

Speed in knots

Definition at line 60 of file telemetry_manager.h.

7.25.3.14 course

```
std::string TelemetryRecord::course
```

Course in degrees

Definition at line 61 of file telemetry_manager.h.

7.25.3.15 date

```
std::string TelemetryRecord::date
```

Date from GPS

Definition at line 62 of file telemetry_manager.h.

7.25.3.16 fix_quality

```
std::string TelemetryRecord::fix_quality
```

GPS fix quality

Definition at line 65 of file telemetry_manager.h.

7.25.3.17 satellites

```
std::string TelemetryRecord::satellites
```

Number of satellites in view

Definition at line 66 of file telemetry_manager.h.

7.25.3.18 altitude

```
std::string TelemetryRecord::altitude
```

Altitude in meters

Definition at line 67 of file telemetry_manager.h.

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

- lib/telemetry/telemetry_manager.h

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 427

8.1.1 Macro Definition Documentation

8.1.1.1 BUILD_NUMBER

```
#define BUILD_NUMBER 427
```

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 427
00007
00008 #endif
```

8.3 includes.h File Reference

```
#include <stdio.h>
#include "pico/stdlib.h"
#include "hardware/spi.h"
#include "hardware/i2c.h"
#include "hardware/uart.h"
#include "pico/multicore.h"
#include "event_manager.h"
#include "lib/powerman/PowerManager.h"
#include <pico/bootrom.h>
#include "ISensor.h"
#include "lib/sensors/BH1750/BH1750_WRAPPER.h"
#include "lib/sensors/BME280/BME280_WRAPPER.h"
#include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
#include "lib/sensors/MPU6050/MPU6050_WRAPPER.h"
#include "lib/clock/DS3231.h"
#include <iostream>
#include <iomanip>
#include <queue>
#include <chrono>
#include "protocol.h"
#include <atomic>
#include <map>
#include "pin_config.h"
#include "utils.h"
#include "communication.h"
#include "build_number.h"
#include "lib/location/gps_collector.h"
#include "lib/storage/storage.h"
#include "lib/storage/pico-vfs/include/filesystem/vfs.h"
#include "telemetry_manager.h"
```

Include dependency graph for includes.h:

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

8.4 includes.h

Go to the documentation of this file.

```
#ifndef INCLUDES_H
#define INCLUDES_H

#include <stdio.h>
#include "pico/stdlib.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" // Corrected path
#include <pico/bootrom.h>

#include "ISensor.h"
#include "lib/sensors/BH1750/BH1750_WRAPPER.h" // Corrected path
#include "lib/sensors/BME280/BME280_WRAPPER.h" // Corrected path
#include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h" // Corrected path
#include "lib/sensors/MPU6050/MPU6050_WRAPPER.h" // Corrected path
#include "lib/clock/DS3231.h" // Corrected path
#include <iostream>
#include <iomanip>
#include <queue>
#include <chrono>
#include "protocol.h"
#include <atomic>
#include <iostream>
#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" // Corrected path
#include "lib/storage/pico-vfs/include/filesystem/vfs.h" // Corrected path
#include "telemetry_manager.h"

#endif
```

8.5 lib/clock/DS3231.cpp File Reference

```
#include "DS3231.h"
#include "utils.h"
#include <stdio.h>
#include <mutex>
#include "event_manager.h"
#include "NMEA_data.h"
```

Include dependency graph for DS3231.cpp:

8.6 DS3231.cpp

Go to the documentation of this file.

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

```
#include "event_manager.h"
#include "NMEA_data.h"

DS3231::DS3231(i2c_inst_t *i2c_instance) : i2c(i2c_instance), ds3231_addr(DS3231_DEVICE_ADRESS) {
    recursive_mutex_init(&clock_mutex_);
}


int DS3231::set_time(ds3231_data_t *data) {
    uint8_t temp[7] = {0};

    if (clock_enable() != 0) {
        uart_print("Failed to enable clock oscillator", VerbosityLevel::ERROR);
        return -1;
    }

    if (data->seconds > 59)
        data->seconds = 59;
    if (data->minutes > 59)
        data->minutes = 59;
    if (data->hours > 23)
        data->hours = 23;
    if (data->day > 7)
        data->day = 7;
    else if (data->day < 1)
        data->day = 1;
    if (data->date > 31)
        data->date = 31;
    else if (data->date < 1)
        data->date = 1;
    if (data->month > 12)
        data->month = 12;
    else if (data->month < 1)
        data->month = 1;
    if (data->year > 99)
        data->year = 99;

    temp[0] = bin_to_bcd(data->seconds);
    temp[1] = bin_to_bcd(data->minutes);
    temp[2] = bin_to_bcd(data->hours);
    temp[2] &= ~(0x01 « 6); // Clear 12/24 hour bit
    temp[3] = bin_to_bcd(data->day);
    temp[4] = bin_to_bcd(data->date);
    temp[5] = bin_to_bcd(data->month);
    if (data->century)
        temp[5] |= (0x01 « 7);
    temp[6] = bin_to_bcd(data->year);

    std::string status = "BCD values to be written to DS3231: " + std::to_string(temp[0]) + " " +
                         std::to_string(temp[1]) + " " + std::to_string(temp[2]) + " " +
                         std::to_string(temp[3]) + " " + std::to_string(temp[4]) + " " +
                         std::to_string(temp[5]) + " " + std::to_string(temp[6]);

    uart_print(status, VerbosityLevel::DEBUG);

    int result = i2c_write_reg(DS3231_SECONDS_REG, 7, temp);
    if (result != 0) {
        uart_print("i2c write failed", VerbosityLevel::ERROR);
        return -1;
    }

    return 0;
}


int DS3231::get_time(ds3231_data_t *data) {
    std::string status;
    uint8_t raw_data[7];
    int result = i2c_read_reg(DS3231_SECONDS_REG, 7, raw_data);
    if (result != 0) {
        status = "Failed to read time from DS3231";
        uart_print(status, VerbosityLevel::ERROR);
        return -1;
    }

    data->seconds = bcd_to_bin(raw_data[0] & 0x7F); // Masking for CH bit (clock halt)
    data->minutes = bcd_to_bin(raw_data[1] & 0x7F);
    data->hours = bcd_to_bin(raw_data[2] & 0x3F);   // Masking for 12/24 hour mode bit
    data->day = raw_data[3] & 0x07;                  // Day of week (1-7)
    data->date = bcd_to_bin(raw_data[4] & 0x3F);
    data->month = bcd_to_bin(raw_data[5] & 0x1F);   // Masking for century bit
    data->century = (raw_data[5] & 0x80) » 7;
    data->year = bcd_to_bin(raw_data[6]);

    if (data->seconds > 59 || data->minutes > 59 || data->hours > 23 ||
        data->day < 1 || data->day > 7 || data->date < 1 || data->date > 31 ||
        data->month < 1 || data->month > 12 || data->year > 99) {
```

```
        uart_print("Invalid data read from DS3231", VerbosityLevel::ERROR);
        return -1;
    }

    return 0;
}


int DS3231::read_temperature(float *resolution) {
    std::string status;
    uint8_t temp[2];
    int result = i2c_read_reg(DS3231_TEMPERATURE_MSB_REG, 2, temp);
    if (result != 0) {
        status = "Failed to read temperature from DS3231";
        uart_print(status, VerbosityLevel::ERROR);
        return -1;
    }

    int8_t temperature_msb = (int8_t)temp[0];
    uint8_t temperature_lsb = temp[1] >> 6;    // Only the 2 MSB are valid

    *resolution = temperature_msb + (temperature_lsb * 0.25f); // 0.25 degree resolution

    return 0;
}


int DS3231::set_unix_time(time_t unix_time) {
    struct tm *timeinfo = gmtime(&unix_time);
    if (timeinfo == NULL) {
        uart_print("Error: gmtime() failed", VerbosityLevel::ERROR);
        return -1;
    }

    ds3231_data_t data;
    data.seconds = timeinfo->tm_sec;
    data.minutes = timeinfo->tm_min;
    data.hours = timeinfo->tm_hour;
    data.day = timeinfo->tm_wday == 0 ? 7 : timeinfo->tm_wday; // Sunday is 0 in tm struct, but 1 in
DS3231
    data.date = timeinfo->tm_mday;
    data.month = timeinfo->tm_mon + 1; // Month is 0-11 in tm struct, but 1-12 in DS3231
    data.year = timeinfo->tm_year - 100; // Year is since 1900, we want the last two digits
    data.century = timeinfo->tm_year >= 2000;

    return set_time(&data);
}


time_t DS3231::get_unix_time() {
    ds3231_data_t data;
    if (get_time(&data)) {
        return -1;
    }

    struct tm timeinfo;
    timeinfo.tm_sec = data.seconds;
    timeinfo.tm_min = data.minutes;
    timeinfo.tm_hour = data.hours;
    timeinfo.tm_mday = data.date;
    timeinfo.tm_mon = data.month - 1; // Month is 0-11 in tm struct, but 1-12 in DS3231
    timeinfo.tm_year = data.year + 100; // Year is since 1900

    // mktime assumes that tm_wday and tm_yday are uninitialized
    timeinfo.tm_wday = 0;
    timeinfo.tm_yday = 0;
    timeinfo.tm_isdst = 0; // Set to 0 to use UTC

    time_t timestamp = mktime(&timeinfo);
    if (timestamp == (time_t)(-1)) {
        uart_print("Error: mktime() failed", VerbosityLevel::ERROR);
        return -1;
    }

    return timestamp;
}


int DS3231::clock_enable() {
    std::string status;
    uint8_t control_reg = 0;
    int result = i2c_read_reg(DS3231_CONTROL_REG, 1, &control_reg);
    if (result != 0) {
        status = "Failed to read control register";
        uart_print(status, VerbosityLevel::ERROR);
        return -1;
    }
```

```

    // Clear the EOSC bit to enable the oscillator
    control_reg &= ~(1 << 7);

    result = i2c_write_reg(DS3231_CONTROL_REG, 1, &control_reg);
    if (result != 0) {
        status = "Failed to write control register";
        uart_print(status, VerbosityLevel::ERROR);
        return -1;
    }

    return 0;
}


int16_t DS3231::get_timezone_offset() const {
    return timezone_offset_minutes_;
}


void DS3231::set_timezone_offset(int16_t offset_minutes) {
    // Validate range: -12 hours to +12 hours (-720 to +720 minutes)
    if (offset_minutes >= -720 && offset_minutes <= 720) {
        timezone_offset_minutes_ = offset_minutes;
    } else {
        uart_print("Error: Invalid timezone offset", VerbosityLevel::ERROR);
    }
}


uint32_t DS3231::get_clock_sync_interval() const {
    return sync_interval_minutes_;
}


void DS3231::set_clock_sync_interval(uint32_t interval_minutes) {
    if (interval_minutes >= 1 && interval_minutes <= 43200) {
        sync_interval_minutes_ = interval_minutes;
    } else {
        uart_print("Error: Invalid sync interval", VerbosityLevel::ERROR);
    }
}


time_t DS3231::get_last_sync_time() const {
    return last_sync_time_;
}


void DS3231::update_last_sync_time() {
    last_sync_time_ = get_unix_time();
    uart_print("Clock sync time updated: " + std::to_string(last_sync_time_), VerbosityLevel::INFO);
}


time_t DS3231::get_local_time() {
    time_t utc_time = get_unix_time();
    if (utc_time == -1) {
        return -1;
    }

    return utc_time + (timezone_offset_minutes_ * 60);
}


bool DS3231::is_sync_needed() {
    if (last_sync_time_ == 0) {
        return true;
    }

    time_t current_time = get_unix_time();
    if (current_time == -1) {
        return true;
    }

    time_t time_since_last_sync = current_time - last_sync_time_;
    uint32_t minutes_since_last_sync = time_since_last_sync / 60;

    return minutes_since_last_sync >= sync_interval_minutes_;
}


bool DS3231::sync_clock_with_gps() {
    extern NMEAData nmea_data;

    if (!nmea_data.has_valid_time()) {
        uart_print("GPS time data not available for sync", VerbosityLevel::WARNING);
```

```
        EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC_DATA_NOT_READY);
        return false;
    }

    time_t gps_time = nmea_data.get_unix_time();
    if (gps_time <= 0) {
        uart_print("Invalid GPS time for sync", VerbosityLevel::ERROR);
        EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC_DATA_NOT_READY);
        return false;
    }

    if (set_unix_time(gps_time) != 0) {
        uart_print("Failed to set system time from GPS", VerbosityLevel::ERROR);
        EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC_DATA_NOT_READY);
        return false;
    }

    update_last_sync_time();

    EventEmitter::emit(EventGroup::CLOCK, ClockEvent::GPS_SYNC);
    uart_print("Clock synced with GPS time: " + std::to_string(gps_time), VerbosityLevel::INFO);

    return true;
}

// ==================== private methods

int DS3231::i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
    if (!length)
        return -1;

    std::string status = "Reading register " + std::to_string(reg_addr) + " from DS3231";
    uart_print(status, VerbosityLevel::DEBUG);
    recursive_mutex_enter_blocking(&clock_mutex_);
    uint8_t reg = reg_addr;
    int write_result = i2c_write_blocking(i2c, ds3231_addr, &reg, 1, true);
    if (write_result == PICO_ERROR_GENERIC) {
        status = "Failed to write register address to DS3231";
        uart_print(status, VerbosityLevel::ERROR);
        recursive_mutex_exit(&clock_mutex_);
        return -1;
    }
    int read_result = i2c_read_blocking(i2c, ds3231_addr, data, length, false);
    if (read_result == PICO_ERROR_GENERIC) {
        status = "Failed to read register data from DS3231";
        uart_print(status, VerbosityLevel::ERROR);
        recursive_mutex_exit(&clock_mutex_);
        return -1;
    }
    recursive_mutex_exit(&clock_mutex_);

    return 0;
}

int DS3231::i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
    if (!length)
        return -1;

    recursive_mutex_enter_blocking(&clock_mutex_);
    uint8_t message[length + 1];
    message[0] = reg_addr;
    for (int i = 0; i < length; i++) {
        message[i + 1] = data[i];
    }
    int write_result = i2c_write_blocking(i2c, ds3231_addr, message, (length + 1), false);
    if (write_result == PICO_ERROR_GENERIC) {
        uart_print("Error: i2c_write_blocking failed in i2c_write_reg", VerbosityLevel::ERROR);
        recursive_mutex_exit(&clock_mutex_);
        return -1;
    }
    recursive_mutex_exit(&clock_mutex_);

    return 0;
}

uint8_t DS3231::bin_to_bcd(const uint8_t data) {
    uint8_t ones_digit = (uint8_t)(data % 10);
    uint8_t tens_digit = (uint8_t)(data - ones_digit) / 10;
    return ((tens_digit << 4) + ones_digit);
}


uint8_t DS3231::bcd_to_bin(const uint8_t bcd) {
    uint8_t ones_digit = (uint8_t)(bcd & 0x0F);
    uint8_t tens_digit = (uint8_t)(bcd >> 4);
    return (tens_digit * 10 + ones_digit);
} // End of DS3231_RTC group
```

8.7 lib/clock/DS3231.h File Reference

```
#include <string>
#include <array>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include <time.h>
#include "pico/mutex.h"
#include "lib/location/gps_collector.h"
```

Include dependency graph for DS3231.h:

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

Classes

- struct ds3231_data_t

  Structure to hold time and date information from DS3231.
- class DS3231

  Class for interfacing with the DS3231 real-time clock.

Macros

- #define DS3231_DEVICE_ADRESS 0x68

  DS3231 I2C device address.
- #define DS3231_SECONDS_REG 0x00

  Register address: Seconds (0-59)
- #define DS3231_MINUTES_REG 0x01

Register address: Minutes (0-59)

- #define DS3231_HOURS_REG 0x02

  Register address: Hours (0-23 in 24hr mode)

- #define DS3231_DAY_REG 0x03

  Register address: Day of the week (1-7)

- #define DS3231_DATE_REG 0x04

  Register address: Date (1-31)

- #define DS3231_MONTH_REG 0x05

  Register address: Month (1-12) & Century bit.

- #define DS3231_YEAR_REG 0x06

  Register address: Year (00-99)

- #define DS3231_CONTROL_REG 0x0E

  Register address: Control register.

- #define DS3231_CONTROL_STATUS_REG 0x0F

  Register address: Control/Status register.

- #define DS3231_TEMPERATURE_MSB_REG 0x11

  Register address: Temperature register (MSB)

- #define DS3231_TEMPERATURE_LSB_REG 0x12

  Register address: Temperature register (LSB)

Enumerations

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

  Enumeration of days of the week.

8.7.1 Macro Definition Documentation

8.7.1.1 DS3231_DEVICE_ADRESS

```
#define DS3231_DEVICE_ADRESS 0x68
```

DS3231 I2C device address.

Definition at line 15 of file DS3231.h.

8.7.1.2 DS3231_SECONDS_REG

```
#define DS3231_SECONDS_REG 0x00
```

Register address: Seconds (0-59)

Definition at line 20 of file DS3231.h.

8.7.1.3 DS3231_MINUTES_REG

```
#define DS3231_MINUTES_REG 0x01
```

Register address: Minutes (0-59)

Definition at line 25 of file DS3231.h.

8.7.1.4 DS3231_HOURS_REG

```
#define DS3231_HOURS_REG 0x02
```

Register address: Hours (0-23 in 24hr mode)

Definition at line 30 of file DS3231.h.

8.7.1.5 DS3231_DAY_REG

```
#define DS3231_DAY_REG 0x03
```

Register address: Day of the week (1-7)

Definition at line 35 of file DS3231.h.

8.7.1.6 DS3231_DATE_REG

```
#define DS3231_DATE_REG 0x04
```

Register address: Date (1-31)

Definition at line 40 of file DS3231.h.

8.7.1.7 DS3231_MONTH_REG

```
#define DS3231_MONTH_REG 0x05
```

Register address: Month (1-12) & Century bit.

Definition at line 45 of file DS3231.h.

8.7.1.8 DS3231_YEAR_REG

```
#define DS3231_YEAR_REG 0x06
```

Register address: Year (00-99)

Definition at line 50 of file DS3231.h.

8.7.1.9 DS3231_CONTROL_REG

```
#define DS3231_CONTROL_REG 0x0E
```

Register address: Control register.

Definition at line 55 of file DS3231.h.

8.7.1.10 DS3231_CONTROL_STATUS_REG

```
#define DS3231_CONTROL_STATUS_REG 0x0F
```

Register address: Control/Status register.

Definition at line 60 of file DS3231.h.

8.7.1.11 DS3231_TEMPERATURE_MSB_REG

```
#define DS3231_TEMPERATURE_MSB_REG 0x11
```

Register address: Temperature register (MSB)

Definition at line 65 of file DS3231.h.

8.7.1.12 DS3231_TEMPERATURE_LSB_REG

```
#define DS3231_TEMPERATURE_LSB_REG 0x12
```

Register address: Temperature register (LSB)

Definition at line 70 of file DS3231.h.

8.7.2 Enumeration Type Documentation

8.7.2.1 days_of_week

```
enum days_of_week
```

Enumeration of days of the week.

Enumerator

MONDAY	Monday.
TUESDAY	Tuesday.
WEDNESDAY	Wednesday.
THURSDAY	Thursday.
FRIDAY	Friday.
SATURDAY	Saturday.
SUNDAY	Sunday.

Definition at line 76 of file DS3231.h.

8.8 DS3231.h

Go to the documentation of this file.

```
#ifndef DS3231_H
#define DS3231_H

#include <string>
#include <array>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include <time.h>
#include "pico/mutex.h"
#include "lib/location/gps_collector.h"

#define DS3231_DEVICE_ADRESS            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

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

typedef struct {
    uint8_t seconds;
    uint8_t minutes;
    uint8_t hours;
    uint8_t day;
    uint8_t date;
    uint8_t month;
    uint8_t year;
    bool century;
} ds3231_data_t;

class DS3231 {
public:
    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();

    int16_t get_timezone_offset() const;

    void set_timezone_offset(int16_t offset_minutes);

    uint32_t get_clock_sync_interval() const;

    void set_clock_sync_interval(uint32_t interval_minutes);

    time_t get_last_sync_time() const;

```

```
    void update_last_sync_time();

    time_t get_local_time();

    bool is_sync_needed();

    bool sync_clock_with_gps();


private:
    i2c_inst_t *i2c;
    uint8_t ds3231_addr;

    int i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data);

    int i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data);

    uint8_t bin_to_bcd(const uint8_t data);

    uint8_t bcd_to_bin(const uint8_t bcd);

    recursive_mutex_t clock_mutex_;
    int16_t timezone_offset_minutes_ = 60;
    uint32_t sync_interval_minutes_ = 1440;
    time_t last_sync_time_ = 0;
};

#endif // DS3231_H
```

8.9 lib/comms/commands/clock_commands.cpp File Reference

```
#include "communication.h"
#include <time.h>
#include "DS3231.h"
```

Include dependency graph for clock_commands.cpp:

Macros

- #define CLOCK_GROUP 3
- #define TIME 0
- #define TIMEZONE_OFFSET 1
- #define CLOCK_SYNC_INTERVAL 2
- #define LAST_SYNC_TIME 3

Functions

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

  Handler for getting and setting system time.
- std::vector< Frame > handle_timezone_offset (const std::string &param, OperationType operationType)

  Handler for getting and setting timezone offset.
- std::vector< Frame > handle_clock_sync_interval (const std::string &param, OperationType operationType)

  Handler for getting and setting clock synchronization interval.
- std::vector< Frame > handle_get_last_sync_time (const std::string &param, OperationType operationType)

  Handler for getting last clock sync time.

Variables

- DS3231 systemClock

8.9.1 Macro Definition Documentation

8.9.1.1 CLOCK_GROUP

```
#define CLOCK_GROUP 3
```

Definition at line 5 of file clock_commands.cpp.

8.9.1.2 TIME

```
#define TIME 0
```

Definition at line 6 of file clock_commands.cpp.

8.9.1.3 TIMEZONE_OFFSET

```
#define TIMEZONE_OFFSET 1
```

Definition at line 7 of file clock_commands.cpp.

8.9.1.4 CLOCK_SYNC_INTERVAL

```
#define CLOCK_SYNC_INTERVAL 2
```

Definition at line 8 of file clock_commands.cpp.

8.9.1.5 LAST_SYNC_TIME

```
#define LAST_SYNC_TIME 3
```

Definition at line 9 of file clock_commands.cpp.

8.10 clock_commands.cpp

Go to the documentation of this file.

```
#include "communication.h"
#include <time.h>
#include "DS3231.h" // Include the DS3231 header

#define CLOCK_GROUP 3
#define TIME 0
#define TIMEZONE_OFFSET 1
#define CLOCK_SYNC_INTERVAL 2
#define LAST_SYNC_TIME 3


 extern DS3231 systemClock;

std::vector<Frame> handle_time(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType == OperationType::SET) {
        if (param.empty()) {
            error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
            return frames;
        }
        try {
            time_t newTime = std::stoll(param);
            if (newTime <= 0) {
                error_msg = error_code_to_string(ErrorCode::INVALID_VALUE);
                frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
                return frames;
            }

            if (systemClock.set_unix_time(newTime) != 0) {
                error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET);
                frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
                return frames;
            }

            EventEmitter::emit(EventGroup::CLOCK, ClockEvent::CHANGED);
            frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, TIME,
std::to_string(systemClock.get_unix_time())));
            return frames;
        } catch (...) {
            error_msg = error_code_to_string(ErrorCode::INVALID_FORMAT);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
            return frames;
        }
    } else if (operationType == OperationType::GET) {
        if (!param.empty()) {
            error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
            return frames;
        }

        uint32_t time_unix = systemClock.get_local_time();
        if (time_unix == 0) {
            error_msg = error_code_to_string(ErrorCode::INTERNAL_FAIL_TO_READ);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
            return frames;
        }

        frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, TIME,
std::to_string(time_unix)));
        return frames;
    }

    error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
    frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIME, error_msg));
    return frames;
}

std::vector<Frame> handle_timezone_offset(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
        return frames;
    }

    if (operationType == OperationType::GET) {
        if (!param.empty()) {
```

```
            error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
            return frames;
        }

        extern DS3231 systemclock;
        int offset = systemClock.get_timezone_offset();
        std::string offset_set = std::to_string(offset);
        frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, TIMEZONE_OFFSET, offset_set));
        return frames;
    }

    if (param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
        frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
        return frames;
    }

    try {
        int16_t offset = std::stoi(param);
        if (offset < -720 || offset > 720) {
            error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
            return frames;
        }

        extern DS3231 systemClock;
        systemClock.set_timezone_offset(offset);
        std::string offset_set = std::to_string(offset);
        frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, TIMEZONE_OFFSET, offset_set));
        return frames;
    } catch (...) {
        error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
        frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, TIMEZONE_OFFSET, error_msg));
        return frames;
    }
}


std::vector<Frame> handle_clock_sync_interval(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, error_msg));
        return frames;
    }

    if (operationType == OperationType::GET) {
        if (!param.empty()) {
            error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, error_msg));
            return frames;
        }

        extern DS3231 systemClock;
        uint32_t syncInterval = systemClock.get_clock_sync_interval();
        std::string clockSyncInterval = std::to_string(syncInterval);
        frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, clockSyncInterval));
        return frames;
    }

    if (operationType == OperationType::SET) {
        if (param.empty()) {
            error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, error_msg));
            return frames;
        }
        try {
            uint32_t interval = std::stoul(param);

            extern DS3231 systemClock;
            systemClock.set_clock_sync_interval(interval);
            std::string interval_set = std::to_string(interval);

            frames.push_back(frame_build(OperationType::RES, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, interval_set));
            return frames;
        } catch (...) {
            error_msg = error_code_to_string(ErrorCode::INVALID_VALUE);
```

```
            frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, error_msg));
        }
    }
    error_msg = error_code_to_string(ErrorCode::UNKNOWN_ERROR);
    frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, error_msg));
    return frames;
}

std::vector<Frame> handle_get_last_sync_time(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET || !param.empty()) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, CLOCK_GROUP, LAST_SYNC_TIME, error_msg));
        return frames;
    }

    extern DS3231 systemClock;
    time_t lastSyncTime = systemClock.get_last_sync_time();

    if (lastSyncTime == 0) {
        frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, LAST_SYNC_TIME, "NEVER"));
    } else {
        frames.push_back(frame_build(OperationType::VAL, CLOCK_GROUP, LAST_SYNC_TIME,
                         std::to_string(lastSyncTime)));
    }

    return frames;
} // end of ClockCommands group
```

8.11 lib/comms/commands/commands.cpp File Reference

```
#include "commands.h"
#include "communication.h"
```

Include dependency graph for commands.cpp:

Typedefs

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

  Function type for command handlers.
- using CommandMap = std::map<uint32_t, CommandHandler>

  Map type for storing command handlers.

Functions

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

  Executes a command based on its key.

Variables

- CommandMap command_handlers

  Global map of all command handlers.

8.12 commands.cpp

Go to the documentation of this file.

```
// commands/commands.cpp
#include "commands.h"
#include "communication.h"


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

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

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


std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType
operationType) {
    auto it = command_handlers.find(commandKey);
    if (it != command_handlers.end()) {
        CommandHandler handler = it->second;
        return handler(param, operationType);
    } else {
        std::vector<Frame> frames;
        frames.push_back(frame_build(OperationType::ERR, 0, 0, "INVALID COMMAND"));
        return frames;
    }
} // end of CommandSystem group
```

8.13 lib/comms/commands/commands.h File Reference

```
#include <string>
#include <functional>
#include <map>
#include "protocol.h"
```

Include dependency graph for commands.h:

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

Functions

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

  Handler for getting and setting system time.
- std::vector< Frame > handle_timezone_offset (const std::string &param, OperationType operationType)

  Handler for getting and setting timezone offset.
- std::vector< Frame > handle_clock_sync_interval (const std::string &param, OperationType operationType)

  Handler for getting and setting clock synchronization interval.
- std::vector< Frame > handle_get_last_sync_time (const std::string &param, OperationType operationType)

  Handler for getting last clock sync time.
- std::vector< Frame > handle_get_commands_list (const std::string &param, OperationType operationType)

  Handler for listing all available commands on UART.
- std::vector< Frame > handle_get_build_version (const std::string &param, OperationType operationType)

  Get firmware build version.
- std::vector< Frame > handle_verbosity (const std::string &param, OperationType operationType)

  Handles setting or getting the UART verbosity level.
- std::vector< Frame > handle_enter_bootloader_mode (const std::string &param, OperationType operation↩ Type)

  Reboot system to USB firmware loader.
- std::vector< Frame > handle_gps_power_status (const std::string &param, OperationType operationType)

  Handler for controlling GPS module power state.
- std::vector< Frame > handle_enable_gps_uart_passthrough (const std::string &param, OperationType operationType)

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

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

  Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- std::vector< Frame > handle_get_gga_data (const std::string &param, OperationType operationType)

  Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.
- std::vector< Frame > handle_get_power_manager_ids (const std::string &param, OperationType operationType)

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

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

  Handler for getting 5V rail voltage.
- std::vector< Frame > handle_get_current_charge_usb (const std::string &param, OperationType operation↩ Type)

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

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

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

  Handler for getting system current draw.
- std::vector< Frame > handle_get_last_events (const std::string &param, OperationType operationType)

  Handler for retrieving last N events from the event log.
- std::vector< Frame > handle_get_event_count (const std::string &param, OperationType operationType)

  Handler for getting total number of events in the log.
- std::vector< Frame > handle_list_files (const std::string &param, OperationType operationType)

  Handles the list files command.
- std::vector< Frame > handle_mount (const std::string &param, OperationType operationType)

  Handles the SD card mount/unmount command.
- std::vector< Frame > handle_get_sensor_data (const std::string &param, OperationType operationType)

  Handler for reading sensor data.
- std::vector< Frame > handle_sensor_config (const std::string &param, OperationType operationType)

  Handler for configuring sensors.
- std::vector< Frame > handle_get_sensor_list (const std::string &param, OperationType operationType)

  Handler for listing available sensors.
- std::vector< Frame > handle_get_last_telemetry_record (const std::string &param, OperationType operationType)

  Handles the get last record command.
- std::vector< Frame > handle_get_last_sensor_record (const std::string &param, OperationType operation↩ Type)

  Handles the get last sensor record command.
- std::vector< Frame > execute_command (uint32_t commandKey, const std::string &param, OperationType operationType)

  Executes a command based on its key.

Variables

- std::map< uint32_t, std::function< std::vector< Frame >(const std::string &, OperationType)> > command_handlers

  Global map of all command handlers.

8.14 commands.h

Go to the documentation of this file.

```
// commands/commands.h
#ifndef COMMANDS_H
#define COMMANDS_H

#include <string>
#include <functional>
#include <map>
#include "protocol.h"

// CLOCK
std::vector<Frame> handle_time(const std::string& param, OperationType operationType);
std::vector<Frame> handle_timezone_offset(const std::string& param, OperationType operationType);
std::vector<Frame> handle_clock_sync_interval(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_last_sync_time(const std::string& param, OperationType operationType);


// DIAG
std::vector<Frame> handle_get_commands_list(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_build_version(const std::string& param, OperationType operationType);
std::vector<Frame> handle_verbosity(const std::string& param, OperationType operationType);
std::vector<Frame> handle_enter_bootloader_mode(const std::string& param, OperationType operationType);


// GPS
std::vector<Frame> handle_gps_power_status(const std::string& param, OperationType operationType);
std::vector<Frame> handle_enable_gps_uart_passthrough(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_rmc_data(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_gga_data(const std::string& param, OperationType operationType);


// POWER
std::vector<Frame> handle_get_power_manager_ids(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_voltage_battery(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_voltage_5v(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_current_charge_usb(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_current_charge_solar(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_current_charge_total(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_current_draw(const std::string& param, OperationType operationType);


// EVENT
std::vector<Frame> handle_get_last_events(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_event_count(const std::string& param, OperationType operationType);


//STORAGE
std::vector<Frame> handle_list_files(const std::string& param, OperationType operationType);
std::vector<Frame> handle_mount(const std::string& param, OperationType operationType);

// SENSOR
std::vector<Frame> handle_get_sensor_data(const std::string& param, OperationType operationType);
std::vector<Frame> handle_sensor_config(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_sensor_list(const std::string& param, OperationType operationType);


// TELEMETRY
std::vector<Frame> handle_get_last_telemetry_record(const std::string& param, OperationType operationType);
std::vector<Frame> handle_get_last_sensor_record(const std::string& param, OperationType operationType);

std::vector<Frame> execute_command(uint32_t commandKey, const std::string& param, OperationType operationType);
extern std::map<uint32_t, std::function<std::vector<Frame>(const std::string&, OperationType)>» command_handlers;

#endif
```

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

```
#include "communication.h"
#include "commands.h"
```

```
#include "pico/stdlib.h"
#include "pico/bootrom.h"
```

Include dependency graph for diagnostic_commands.cpp:

Functions

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

  Handler for listing all available commands on UART.
- std::vector< Frame > handle_get_build_version (const std::string &param, OperationType operationType)

  Get firmware build version.
- std::vector< Frame > handle_verbosity (const std::string &param, OperationType operationType)

  Handles setting or getting the UART verbosity level.
- std::vector< Frame > handle_enter_bootloader_mode (const std::string &param, OperationType operation↩ Type)

  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/stdlib.h"
00004 #include "pico/bootrom.h"
00005
00010
00011 extern volatile bool g_pending_bootloader_reset;
00012
00023 std::vector<Frame> handle_get_commands_list(const std::string& param, OperationType operationType) {
00024     std::vector<Frame> frames;
00025     std::string error_msg;
00026
00027     if (!param.empty()) {
00028         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00029         frames.push_back(frame_build(OperationType::ERR, 1, 0, error_msg));
00030         return frames;
00031     }
00032
00033     if (!(operationType == OperationType::GET)) {
00034         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00035         frames.push_back(frame_build(OperationType::ERR, 1, 0, error_msg));
00036         return frames;
00037     }
00038
00039     std::string combined_command_details;
00040     for (const auto& entry : command_handlers) {
00041         uint32_t command_key = entry.first;
00042         uint8_t group = (command_key » 8) & 0xFF;
00043         uint8_t command = command_key & 0xFF;
00044
00045         std::string command_details = std::to_string(group) + "." + std::to_string(command);
00046
00047         if (combined_command_details.length() + command_details.length() + 1 > 100) {
```

```
            frames.push_back(frame_build(OperationType::SEQ, 1, 0, combined_command_details));
            combined_command_details = "";
        }

        if (!combined_command_details.empty()) {
            combined_command_details += "-";
        }
        combined_command_details += command_details;
    }

    if (!combined_command_details.empty()) {
        frames.push_back(frame_build(OperationType::SEQ, 1, 0, combined_command_details));
    }

    frames.push_back(frame_build(OperationType::VAL, 1, 0, "SEQ_DONE"));
    return frames;
}


std::vector<Frame> handle_get_build_version(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, 1, 1, error_msg));
        return frames;
    }

    if (operationType == OperationType::GET) {
        frames.push_back(frame_build(OperationType::VAL, 1, 1, std::to_string(BUILD_NUMBER)));
        return frames;
    }

    error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
    frames.push_back(frame_build(OperationType::ERR, 1, 1, error_msg));
    return frames;
}


std::vector<Frame> handle_verbosity(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType == OperationType::GET && param.empty()) {
        uart_print("Current verbosity level: " + std::to_string(static_cast<int>(g_uart_verbosity)),
                   VerbosityLevel::INFO);
        frames.push_back(frame_build(OperationType::VAL, 1, 8,
                        std::to_string(static_cast<int>(g_uart_verbosity))));
        return frames;
    }

    try {
        int level = std::stoi(param);
        if (level < 0 || level > 5) {
            error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
            frames.push_back(frame_build(OperationType::ERR, 1, 8, error_msg));
            return frames;
        }
        g_uart_verbosity = static_cast<VerbosityLevel>(level);
        frames.push_back(frame_build(OperationType::RES, 1, 8, "SET " + std::to_string(level)));
        return frames;
    } catch (...) {
        error_msg = error_code_to_string(ErrorCode::INVALID_FORMAT);
        frames.push_back(frame_build(OperationType::ERR, 1, 8, error_msg));
        return frames;
    }
}

std::vector<Frame> handle_enter_bootloader_mode(const std::string& param, OperationType operationType)
    {
    std::vector<Frame> frames;
    std::string error_msg;

    if (param != "USB") {
        error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
        frames.push_back(frame_build(OperationType::ERR, 1, 9, error_msg));
        return frames;
    }

    if (operationType != OperationType::SET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, 1, 9, error_msg));
        return frames;
    }

    frames.push_back(frame_build(OperationType::RES, 1, 9, "REBOOT BOOTSEL"));
```

```
    // Set flag to trigger reboot after function returns
    g_pending_bootloader_reset = true;

    return frames;
}
```

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

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

Include dependency graph for event_commands.cpp:

Functions

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

  Handler for retrieving last N events from the event log.
- std::vector< Frame > handle_get_event_count (const std::string &param, OperationType operationType)

  Handler for getting total number of events in the log.

8.18 event_commands.cpp

Go to the documentation of this file.

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


std::vector<Frame> handle_get_last_events(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, 5, 1, error_msg));
        return frames;
    }
    size_t count = 10; // Default number of events to return
    if (!param.empty()) {
        try {
            count = std::stoul(param);
            if (count > EVENT_BUFFER_SIZE) {
                error_msg = error_code_to_string(ErrorCode::INVALID_VALUE);
                frames.push_back(frame_build(OperationType::ERR, 5, 1, error_msg));
                return frames;
            }
        } catch (...) {
            error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
            frames.push_back(frame_build(OperationType::ERR, 5, 1, error_msg));
            return frames;
        }
    }

```

```
    size_t available = eventManager.get_event_count();
    size_t to_return = (count == 0) ? available : std::min(count, available);
    size_t event_index = available;

    while (to_return > 0) {
        std::stringstream ss;
        ss « std::hex « std::uppercase « std::setfill('0');
        size_t events_in_frame = 0;

        for (size_t i = 0; i < 10 && to_return > 0; ++i) {
            event_index--;
            const EventLog& event = eventManager.get_event(event_index);

            ss « std::setw(4) « event.id
               « std::setw(8) « event.timestamp
               « std::setw(2) « static_cast<int>(event.group)
               « std::setw(2) « static_cast<int>(event.event);

            if (to_return > 1) ss « "-";

            to_return--;
            events_in_frame++;
        }
        frames.push_back(frame_build(OperationType::SEQ, 5, 1, ss.str()));
    }
    frames.push_back(frame_build(OperationType::VAL, 5, 1, "SEQ_DONE"));
    return frames;
}


std::vector<Frame> handle_get_event_count(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET || !param.empty()) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, 5, 2, error_msg));
        return frames;
    }
    frames.push_back(frame_build(OperationType::VAL, 5, 2,
std::to_string(eventManager.get_event_count())));
    return frames;
} // end of EventCommands group
```

8.19 lib/comms/commands/gps_commands.cpp File Reference

```
#include "communication.h"
#include "lib/location/gps_collector.h"
#include <sstream>
```

Include dependency graph for gps_commands.cpp:

Macros

- #define GPS_GROUP 7
- #define POWER_STATUS_COMMAND 1
- #define PASSTHROUGH_COMMAND 2
- #define RMC_DATA_COMMAND 3
- #define GGA_DATA_COMMAND 4

Functions

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

  Handler for controlling GPS module power state.
- std::vector< Frame > handle_enable_gps_uart_passthrough (const std::string &param, OperationType operationType)

  Handler for enabling GPS transparent mode (UART pass-through)
- std::vector< Frame > handle_get_rmc_data (const std::string &param, OperationType operationType)

  Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- std::vector< Frame > handle_get_gga_data (const std::string &param, OperationType operationType)

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

8.19.1 Macro Definition Documentation

8.19.1.1 GPS_GROUP

```
#define GPS_GROUP 7
```

Definition at line 5 of file gps_commands.cpp.

8.19.1.2 POWER_STATUS_COMMAND

```
#define POWER_STATUS_COMMAND 1
```

Definition at line 6 of file gps_commands.cpp.

8.19.1.3 PASSTHROUGH_COMMAND

```
#define PASSTHROUGH_COMMAND 2
```

Definition at line 7 of file gps_commands.cpp.

8.19.1.4 RMC_DATA_COMMAND

```
#define RMC_DATA_COMMAND 3
```

Definition at line 8 of file gps_commands.cpp.

8.19.1.5 GGA_DATA_COMMAND

```
#define GGA_DATA_COMMAND 4
```

Definition at line 9 of file gps_commands.cpp.

8.20 gps_commands.cpp

Go to the documentation of this file.

```
#include "communication.h"
#include "lib/location/gps_collector.h"
#include <sstream> // Include for stringstream

#define GPS_GROUP 7
#define POWER_STATUS_COMMAND 1
#define PASSTHROUGH_COMMAND 2
#define RMC_DATA_COMMAND 3
#define GGA_DATA_COMMAND 4


std::vector<Frame> handle_gps_power_status(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_str;

    if (operationType == OperationType::SET) {
        if (param.empty()) {
            error_str = error_code_to_string(ErrorCode::PARAM_REQUIRED);
            frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND, error_str));
            return frames;
        }

        try {
            int power_status = std::stoi(param);
            if (power_status != 0 && power_status != 1) {
                error_str = error_code_to_string(ErrorCode::PARAM_INVALID);
                frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND, error_str));
                return frames;
            }
            gpio_put(GPS_POWER_ENABLE_PIN, power_status);
            EventEmitter::emit(EventGroup::GPS, power_status ? GPSEvent::POWER_ON : GPSEvent::POWER_OFF);
            frames.push_back(frame_build(OperationType::RES, GPS_GROUP, POWER_STATUS_COMMAND, std::to_string(power_status)));
            return frames;
        } catch (...) {
            error_str = error_code_to_string(ErrorCode::PARAM_INVALID);
            frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND, error_str));
            return frames;
        }
    }

    // GET operation
    if (!param.empty()) {
        error_str = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, POWER_STATUS_COMMAND, error_str));
        return frames;
    }

    bool power_status = gpio_get(GPS_POWER_ENABLE_PIN);
    frames.push_back(frame_build(OperationType::VAL, GPS_GROUP, POWER_STATUS_COMMAND, std::to_string(power_status)));
    return frames;
}


std::vector<Frame> handle_enable_gps_uart_passthrough(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_str;

    if (!(operationType == OperationType::SET)) {
        error_str = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, PASSTHROUGH_COMMAND, error_str));
        return frames;
    }

    // Parse and validate timeout parameter
    uint32_t timeout_ms;
    try {
        timeout_ms = param.empty() ? 60000u : std::stoul(param) * 1000;
    } catch (...) {
        error_str = error_code_to_string(ErrorCode::INVALID_VALUE);
        frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, PASSTHROUGH_COMMAND, error_str));
        return frames;
    }

    // Setup UART parameters and exit sequence
    const std::string EXIT_SEQUENCE = "##EXIT##";
```

```
    std::string input_buffer;
    bool exit_requested = false;
    static volatile bool pause_gps_collection = false;
    pause_gps_collection = true;
    sleep_ms(100);

    uint32_t original_baud_rate = DEBUG_UART_BAUD_RATE;
    uint32_t gps_baud_rate = GPS_UART_BAUD_RATE;
    uint32_t start_time = to_ms_since_boot(get_absolute_time());

    EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_START);

    std::string message = "Entering GPS Serial Pass-Through Mode @" +
                          std::to_string(gps_baud_rate) + " for " +
                          std::to_string(timeout_ms/1000) + "s\r\n" +
                          "Send " + EXIT_SEQUENCE + " to exit";
    uart_print(message, VerbosityLevel::INFO);

    sleep_ms(10);

    uart_set_baudrate(DEBUG_UART_PORT, gps_baud_rate);

    while (!exit_requested) {
        while (uart_is_readable(DEBUG_UART_PORT)) {
            char ch = uart_getc(DEBUG_UART_PORT);

            input_buffer += ch;
            if (input_buffer.length() > EXIT_SEQUENCE.length()) {
                input_buffer = input_buffer.substr(1);
            }

            if (input_buffer == EXIT_SEQUENCE) {
                exit_requested = true;
                break;
            }

            if (input_buffer != EXIT_SEQUENCE.substr(0, input_buffer.length())) {
                uart_write_blocking(GPS_UART_PORT,
                    reinterpret_cast<const uint8_t*>(&ch), 1);
            }
        }

        while (uart_is_readable(GPS_UART_PORT)) {
            char gps_byte = uart_getc(GPS_UART_PORT);
            uart_write_blocking(DEBUG_UART_PORT,
                reinterpret_cast<const uint8_t*>(&gps_byte), 1);
        }

        if (to_ms_since_boot(get_absolute_time()) - start_time >= timeout_ms) {
            break;
        }
    }

    uart_set_baudrate(DEBUG_UART_PORT, original_baud_rate);

    sleep_ms(50);

    pause_gps_collection = false;

    EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_END);

    std::string exit_reason = exit_requested ? "USER_EXIT" : "TIMEOUT";
    std::string response = "GPS UART BRIDGE EXIT: " + exit_reason;
    uart_print(response, VerbosityLevel::INFO);

    frames.push_back(frame_build(OperationType::RES, GPS_GROUP, PASSTHROUGH_COMMAND, response));
    return frames;
}


std::vector<Frame> handle_get_rmc_data(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, RMC_DATA_COMMAND, error_msg));
        return frames;
    }

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, RMC_DATA_COMMAND, error_msg));
        return frames;
    }

    std::vector<std::string> tokens = nmea_data.get_rmc_tokens();
```

```
    std::stringstream ss;
    for (size_t i = 0; i < tokens.size(); ++i) {
        ss « tokens[i];
        if (i < tokens.size() - 1) {
            ss « ",";
        }
    }

    frames.push_back(frame_build(OperationType::VAL, GPS_GROUP, RMC_DATA_COMMAND, ss.str()));
    return frames;
}


std::vector<Frame> handle_get_gga_data(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, GGA_DATA_COMMAND, error_msg));
        return frames;
    }

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, GPS_GROUP, GGA_DATA_COMMAND, error_msg));
        return frames;
    }

    std::vector<std::string> tokens = nmea_data.get_gga_tokens();
    std::stringstream ss;
    for (size_t i = 0; i < tokens.size(); ++i) {
        ss « tokens[i];
        if (i < tokens.size() - 1) {
            ss « ",";
        }
    }

    frames.push_back(frame_build(OperationType::VAL, GPS_GROUP, GGA_DATA_COMMAND, ss.str()));
    return frames;
} // end of GPSCommands group
```

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

`#include "communication.h"`

Include dependency graph for power_commands.cpp:

Macros

- #define POWER_GROUP 2
- #define POWER_MANAGER_IDS 0
- #define VOLTAGE_BATTERY 2
- #define VOLTAGE_MAIN 3
- #define CHARGE_USB 4
- #define CHARGE_SOLAR 5
- #define CHARGE_TOTAL 6
- #define DRAW_TOTAL 7

Functions

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

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

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

  Handler for getting 5V rail voltage.
- std::vector< Frame > handle_get_current_charge_usb (const std::string &param, OperationType operation↩ Type)

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

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

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

  Handler for getting system current draw.

8.21.1 Macro Definition Documentation

8.21.1.1 POWER_GROUP

```
#define POWER_GROUP 2
```

Definition at line 3 of file power_commands.cpp.

8.21.1.2 POWER_MANAGER_IDS

```
#define POWER_MANAGER_IDS 0
```

Definition at line 4 of file power_commands.cpp.

8.21.1.3 VOLTAGE_BATTERY

```
#define VOLTAGE_BATTERY 2
```

Definition at line 5 of file power_commands.cpp.

8.21.1.4 VOLTAGE_MAIN

```
#define VOLTAGE_MAIN 3
```

Definition at line 6 of file power_commands.cpp.

8.21.1.5 CHARGE_USB

```
#define CHARGE_USB 4
```

Definition at line 7 of file power_commands.cpp.

8.21.1.6 CHARGE_SOLAR

```
#define CHARGE_SOLAR 5
```

Definition at line 8 of file power_commands.cpp.

8.21.1.7 CHARGE_TOTAL

```
#define CHARGE_TOTAL 6
```

Definition at line 9 of file power_commands.cpp.

8.21.1.8 DRAW_TOTAL

```
#define DRAW_TOTAL 7
```

Definition at line 10 of file power_commands.cpp.

8.22 power_commands.cpp

Go to the documentation of this file.

```
00001 #include "communication.h"
00002
00003 #define POWER_GROUP 2
00004 #define POWER_MANAGER_IDS 0
00005 #define VOLTAGE_BATTERY 2
00006 #define VOLTAGE_MAIN 3
00007 #define CHARGE_USB 4
00008 #define CHARGE_SOLAR 5
00009 #define CHARGE_TOTAL 6
00010 #define DRAW_TOTAL 7
00011
00017
00030 std::vector<Frame> handle_get_power_manager_ids(const std::string& param, OperationType operationType)
      {
00031     std::vector<Frame> frames;
00032     std::string error_msg;
00033
00034     if (!param.empty()) {
00035         error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
00036         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, POWER_MANAGER_IDS, error_msg));
00037         return frames;
00038     }
00039
00040     if (!(operationType == OperationType::GET)) {
00041         error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
00042         frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, POWER_MANAGER_IDS, error_msg));
00043         return frames;
00044     }
00045
00046     extern PowerManager powerManager;
00047     std::string power_manager_ids = powerManager.read_device_ids();
00048     frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, POWER_MANAGER_IDS,
      power_manager_ids));
00049     return frames;
```

```
}

std::vector<Frame> handle_get_voltage_battery(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_BATTERY, error_msg));
        return frames;
    }

    if (!(operationType == OperationType::GET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_BATTERY, error_msg));
        return frames;
    }

    extern PowerManager powerManager;
    float voltage = powerManager.get_voltage_battery();
    frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, VOLTAGE_BATTERY,
std::to_string(voltage), ValueUnit::VOLT));
    return frames;
}

std::vector<Frame> handle_get_voltage_5v(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_MAIN, error_msg));
        return frames;
    }

    if (!(operationType == OperationType::GET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, VOLTAGE_MAIN, error_msg));
        return frames;
    }

    extern PowerManager powerManager;
    float voltage = powerManager.get_voltage_5v();
    frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, VOLTAGE_MAIN,
std::to_string(voltage), ValueUnit::VOLT));
    return frames;
}

std::vector<Frame> handle_get_current_charge_usb(const std::string& param, OperationType
operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_USB, error_msg));
        return frames;
    }

    if (!(operationType == OperationType::GET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_USB, error_msg));
        return frames;
    }

    extern PowerManager powerManager;
    float charge_current = powerManager.get_current_charge_usb();
    frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, CHARGE_USB,
std::to_string(charge_current), ValueUnit::MILIAMP));
    return frames;
}

std::vector<Frame> handle_get_current_charge_solar(const std::string& param, OperationType
operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_SOLAR, error_msg));
        return frames;
    }

    if (!(operationType == OperationType::GET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_SOLAR, error_msg));
        return frames;
```

```
    }

    extern PowerManager powerManager;
    float charge_current = powerManager.get_current_charge_solar();
    frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, CHARGE_SOLAR,
std::to_string(charge_current), ValueUnit::MILIAMP));
    return frames;
}

std::vector<Frame> handle_get_current_charge_total(const std::string& param, OperationType
operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_TOTAL, error_msg));
        return frames;
    }

    if (!(operationType == OperationType::GET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, CHARGE_TOTAL, error_msg));
        return frames;
    }

    extern PowerManager powerManager;
    float charge_current = powerManager.get_current_charge_total();
    frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, CHARGE_TOTAL,
std::to_string(charge_current), ValueUnit::MILIAMP));
    return frames;
}

std::vector<Frame> handle_get_current_draw(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (!param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_UNNECESSARY);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, DRAW_TOTAL, error_msg));
        return frames;
    }

    if (!(operationType == OperationType::GET)) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, POWER_GROUP, DRAW_TOTAL, error_msg));
        return frames;
    }

    extern PowerManager powerManager;
    float current_draw = powerManager.get_current_draw();
    frames.push_back(frame_build(OperationType::VAL, POWER_GROUP, DRAW_TOTAL,
std::to_string(current_draw), ValueUnit::MILIAMP));
    return frames;
} // end of PowerCommands group
```

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

```
#include "communication.h"
#include "ISensor.h"
#include <vector>
#include <string>
#include <sstream>
#include "commands.h"
```

Include dependency graph for sensor_commands.cpp:

Macros

- #define SENSOR_GROUP 4
- #define SENSOR_READ 0
- #define SENSOR_CONFIGURE 1

Functions

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

  Handler for reading sensor data.
- std::vector< Frame > handle_sensor_config (const std::string &param, OperationType operationType)

  Handler for configuring sensors.
- std::vector< Frame > handle_get_sensor_list (const std::string &param, OperationType operationType)

  Handler for listing available sensors.

8.23.1 Macro Definition Documentation

8.23.1.1 SENSOR_GROUP

```
#define SENSOR_GROUP 4
```

Definition at line 8 of file sensor_commands.cpp.

8.23.1.2 SENSOR_READ

```
#define SENSOR_READ 0
```

Definition at line 9 of file sensor_commands.cpp.

8.23.1.3 SENSOR_CONFIGURE

```
#define SENSOR_CONFIGURE 1
```

Definition at line 10 of file sensor_commands.cpp.

8.24 sensor_commands.cpp

Go to the documentation of this file.

```
#include "communication.h"
#include "ISensor.h"
#include <vector>
#include <string>
#include <sstream>
#include "commands.h"

#define SENSOR_GROUP 4
#define SENSOR_READ 0
#define SENSOR_CONFIGURE 1


std::vector<Frame> handle_get_sensor_data(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
        return frames;
    }

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
        return frames;
    }

    // Parse sensor type and data type from param
    std::string sensor_type_str;
    std::string data_type_str;

    size_t dash_pos = param.find('-');
    if (dash_pos != std::string::npos) {
        sensor_type_str = param.substr(0, dash_pos);
        data_type_str = param.substr(dash_pos + 1);
    } else {
        sensor_type_str = param;
    }

    // Convert sensor_type_str to SensorType
    SensorType sensor_type;
    if (sensor_type_str == "light") {
        sensor_type = SensorType::LIGHT;
    } else if (sensor_type_str == "environment") {
        sensor_type = SensorType::ENVIRONMENT;
    } else if (sensor_type_str == "magnetometer") {
        sensor_type = SensorType::MAGNETOMETER;
    } else if (sensor_type_str == "imu") {
        sensor_type = SensorType::IMU;
    } else {
        error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Invalid sensor type";
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
        return frames;
    }

    SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();

    // If data type is specified, read that specific data
    if (!data_type_str.empty()) {
        SensorDataTypeIdentifier data_type;

        // Map string to SensorDataTypeIdentifier
        if (data_type_str == "light_level") {
            data_type = SensorDataTypeIdentifier::LIGHT_LEVEL;
        } else if (data_type_str == "temperature") {
            data_type = SensorDataTypeIdentifier::TEMPERATURE;
        } else if (data_type_str == "pressure") {
            data_type = SensorDataTypeIdentifier::PRESSURE;
        } else if (data_type_str == "humidity") {
            data_type = SensorDataTypeIdentifier::HUMIDITY;
        } else if (data_type_str == "mag_field_x") {
            data_type = SensorDataTypeIdentifier::MAG_FIELD_X;
        } else if (data_type_str == "mag_field_y") {
            data_type = SensorDataTypeIdentifier::MAG_FIELD_Y;
        } else if (data_type_str == "mag_field_z") {
            data_type = SensorDataTypeIdentifier::MAG_FIELD_Z;
        } else if (data_type_str == "gyro_x") {
            data_type = SensorDataTypeIdentifier::GYRO_X;
        } else if (data_type_str == "gyro_y") {
            data_type = SensorDataTypeIdentifier::GYRO_Y;
        } else if (data_type_str == "gyro_z") {
```

```
            data_type = SensorDataTypeIdentifier::GYRO_Z;
        } else if (data_type_str == "accel_x") {
            data_type = SensorDataTypeIdentifier::ACCEL_X;
        } else if (data_type_str == "accel_y") {
            data_type = SensorDataTypeIdentifier::ACCEL_Y;
        } else if (data_type_str == "accel_z") {
            data_type = SensorDataTypeIdentifier::ACCEL_Z;
        } else {
            error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Invalid data type";
            frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_READ, error_msg));
            return frames;
        }

        float value = sensor_wrapper.sensor_read_data(sensor_type, data_type);
        std::stringstream ss;
        ss « value;
        frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, SENSOR_READ, ss.str()));
    }
    // If only sensor type is specified, read all relevant data for that sensor type
    else {
        std::vector<SensorDataTypeIdentifier> data_types;

        switch (sensor_type) {
            case SensorType::LIGHT:
                data_types = {SensorDataTypeIdentifier::LIGHT_LEVEL};
                break;
            case SensorType::ENVIRONMENT:
                data_types = {
                    SensorDataTypeIdentifier::TEMPERATURE,
                    SensorDataTypeIdentifier::PRESSURE,
                    SensorDataTypeIdentifier::HUMIDITY
                };
                break;
            case SensorType::MAGNETOMETER:
                data_types = {
                    SensorDataTypeIdentifier::MAG_FIELD_X,
                    SensorDataTypeIdentifier::MAG_FIELD_Y,
                    SensorDataTypeIdentifier::MAG_FIELD_Z
                };
                break;
            case SensorType::IMU:
                data_types = {
                    SensorDataTypeIdentifier::GYRO_X,
                    SensorDataTypeIdentifier::GYRO_Y,
                    SensorDataTypeIdentifier::GYRO_Z,
                    SensorDataTypeIdentifier::ACCEL_X,
                    SensorDataTypeIdentifier::ACCEL_Y,
                    SensorDataTypeIdentifier::ACCEL_Z
                };
                break;
        }

        std::stringstream combined_values;
        std::vector<std::string> data_type_names;
        std::vector<float> values;

        // Get names for the data types and store the values
        for (SensorDataTypeIdentifier data_type : data_types) {
            switch (data_type) {
                case SensorDataTypeIdentifier::LIGHT_LEVEL:
                    data_type_names.push_back("light_level");
                    break;
                case SensorDataTypeIdentifier::TEMPERATURE:
                    data_type_names.push_back("temperature");
                    break;
                case SensorDataTypeIdentifier::PRESSURE:
                    data_type_names.push_back("pressure");
                    break;
                case SensorDataTypeIdentifier::HUMIDITY:
                    data_type_names.push_back("humidity");
                    break;
                case SensorDataTypeIdentifier::MAG_FIELD_X:
                    data_type_names.push_back("mag_field_x");
                    break;
                case SensorDataTypeIdentifier::MAG_FIELD_Y:
                    data_type_names.push_back("mag_field_y");
                    break;
                case SensorDataTypeIdentifier::MAG_FIELD_Z:
                    data_type_names.push_back("mag_field_z");
                    break;
                case SensorDataTypeIdentifier::GYRO_X:
                    data_type_names.push_back("gyro_x");
                    break;
                case SensorDataTypeIdentifier::GYRO_Y:
                    data_type_names.push_back("gyro_y");
                    break;
                case SensorDataTypeIdentifier::GYRO_Z:
```

```
                    data_type_names.push_back("gyro_z");
                    break;
                case SensorDataTypeIdentifier::ACCEL_X:
                    data_type_names.push_back("accel_x");
                    break;
                case SensorDataTypeIdentifier::ACCEL_Y:
                    data_type_names.push_back("accel_y");
                    break;
                case SensorDataTypeIdentifier::ACCEL_Z:
                    data_type_names.push_back("accel_z");
                    break;
            }

            float value = sensor_wrapper.sensor_read_data(sensor_type, data_type);
            values.push_back(value);
        }

        // Format output as key-value pairs
        for (size_t i = 0; i < data_type_names.size(); i++) {
            if (i > 0) combined_values « "|";
            combined_values « data_type_names[i] « ":" « values[i];
        }

        frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, SENSOR_READ, combined_values.str()));
    }

    return frames;
}

std::vector<Frame> handle_sensor_config(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (param.empty()) {
        error_msg = error_code_to_string(ErrorCode::PARAM_REQUIRED);
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
        return frames;
    }

    if (operationType != OperationType::SET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
        return frames;
    }

    // Parse sensor type and configuration from param
    size_t semicolon_pos = param.find(';');
    if (semicolon_pos == std::string::npos) {
        error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Format should be sensor_type;config_params";
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
        return frames;
    }

    std::string sensor_type_str = param.substr(0, semicolon_pos);
    std::string config_str = param.substr(semicolon_pos + 1);

    // Convert sensor_type_str to SensorType
    SensorType sensor_type;
    if (sensor_type_str == "light") {
        sensor_type = SensorType::LIGHT;
    } else if (sensor_type_str == "environment") {
        sensor_type = SensorType::ENVIRONMENT;
    } else if (sensor_type_str == "magnetometer") {
        sensor_type = SensorType::MAGNETOMETER;
    } else if (sensor_type_str == "imu") {
        sensor_type = SensorType::IMU;
    } else {
        error_msg = error_code_to_string(ErrorCode::PARAM_INVALID) + ": Invalid sensor type";
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
        return frames;
    }

    // Parse configuration parameters
    std::map<std::string, std::string> config_map;
    std::stringstream ss(config_str);
    std::string config_pair;

    while (std::getline(ss, config_pair, '|')) {
        size_t colon_pos = config_pair.find(':');
        if (colon_pos != std::string::npos) {
            std::string key = config_pair.substr(0, colon_pos);
            std::string value = config_pair.substr(colon_pos + 1);
            config_map[key] = value;
        }
    }
```

```

    // Apply configuration
    SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();
    bool success = sensor_wrapper.sensor_configure(sensor_type, config_map);

    if (success) {
        frames.push_back(frame_build(OperationType::RES, SENSOR_GROUP, SENSOR_CONFIGURE, "Configuration successful"));
    } else {
        error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET) + ": Failed to configure sensor";
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, SENSOR_CONFIGURE, error_msg));
    }

    return frames;
}


std::vector<Frame> handle_get_sensor_list(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, SENSOR_GROUP, 2, error_msg));
        return frames;
    }

    // Get the singleton instance
    SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();

    // Get list of available sensor types
    std::vector<std::pair<SensorType, uint8_t» available_sensors = sensor_wrapper.get_available_sensors();

    if (available_sensors.empty()) {
        frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, 2, "No sensors available"));
        return frames;
    }

    std::stringstream sensor_list;
    bool first = true;

    for (const auto& sensor_info : available_sensors) {
        if (!first) {
            sensor_list « "|";
        }

        // Format: sensor_type:address (in hex)
        std::stringstream addr_hex;
        addr_hex « std::hex « static_cast<int>(sensor_info.second);

        switch (sensor_info.first) {
            case SensorType::LIGHT:
                sensor_list « "light:0x" « addr_hex.str();
                break;
            case SensorType::ENVIRONMENT:
                sensor_list « "environment:0x" « addr_hex.str();
                break;
            case SensorType::MAGNETOMETER:
                sensor_list « "magnetometer:0x" « addr_hex.str();
                break;
            case SensorType::IMU:
                sensor_list « "imu:0x" « addr_hex.str();
                break;
            default:
                sensor_list « "unknown:0x" « addr_hex.str();
                break;
        }

        first = false;
    }

    frames.push_back(frame_build(OperationType::VAL, SENSOR_GROUP, 2, sensor_list.str()));
    return frames;
}
```

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

```
#include "commands.h"
#include "communication.h"
```

```
#include "storage.h"
#include "filesystem/vfs.h"
#include "filesystem/littlefs.h"
#include <sys/stat.h>
#include <errno.h>
#include "dirent.h"
```

Include dependency graph for storage_commands.cpp:

Macros

- #define STORAGE_GROUP 6
- #define LIST_FILES_COMMAND 0
- #define MOUNT_COMMAND 4

Functions

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

  Handles the list files command.
- std::vector< Frame > handle_mount (const std::string &param, OperationType operationType)

  Handles the SD card mount/unmount command.

8.25.1 Macro Definition Documentation

8.25.1.1 STORAGE_GROUP

```
#define STORAGE_GROUP 6
```

Definition at line 10 of file storage_commands.cpp.

8.25.1.2 LIST_FILES_COMMAND

```
#define LIST_FILES_COMMAND 0
```

Definition at line 12 of file storage_commands.cpp.

8.25.1.3 MOUNT_COMMAND

```
#define MOUNT_COMMAND 4
```

Definition at line 13 of file storage_commands.cpp.

8.26 storage_commands.cpp

Go to the documentation of this file.

```
#include "commands.h"
#include "communication.h"
#include "storage.h"
#include "filesystem/vfs.h"
#include "filesystem/littlefs.h"
#include <sys/stat.h>
#include <errno.h>
#include "dirent.h"

#define STORAGE_GROUP 6

#define LIST_FILES_COMMAND 0
#define MOUNT_COMMAND 4

std::vector<Frame> handle_list_files(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND, error_msg));
        return frames;
    }

    DIR* dir;
    struct dirent* ent;
    int file_count = 0; // Counter for the number of files
    if ((dir = opendir("/")) != NULL) {
        // First, count the number of files
        while ((ent = readdir(dir)) != NULL) {
            const char* filename = ent->d_name;
            if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0) {
                continue;
            }
            file_count++;
        }
        closedir(dir);

        // Send the number of files
        frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, LIST_FILES_COMMAND, std::to_string(file_count)));

        // Open the directory again to read file information
        dir = opendir("/");
        if (dir != NULL) {
            while ((ent = readdir(dir)) != NULL) {
                const char* filename = ent->d_name;

                // Skip "." and ".." directories
                if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0) {
                    continue;
                }

                // Get file size
                char filepath[256];
                snprintf(filepath, sizeof(filepath), "/%s", filename);

                FILE* file = fopen(filepath, "rb");
                size_t file_size = 0;

                if (file != NULL) {
                    fseek(file, 0, SEEK_END);
                    file_size = ftell(file);
                    fclose(file);
                }

                // Create and send frame with filename and size
                char file_info[512];
                snprintf(file_info, sizeof(file_info), "%s:%zu", filename, file_size);
                frames.push_back(frame_build(OperationType::SEQ, STORAGE_GROUP, LIST_FILES_COMMAND, file_info));
            }
            closedir(dir);
            frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, LIST_FILES_COMMAND, "SEQ_DONE"));
            return frames;
        } else {
            error_msg = error_code_to_string(ErrorCode::INTERNAL_FAIL_TO_READ);
            frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND, error_msg));
            return frames;
```

```
        }
    } else {
        frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, LIST_FILES_COMMAND, error_msg));
        return frames;
    }
}

std::vector<Frame> handle_mount(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType == OperationType::GET) {
        frames.push_back(frame_build(OperationType::VAL, STORAGE_GROUP, MOUNT_COMMAND, std::to_string(sd_card_mounted)));
        return frames;
    } else if (operationType == OperationType::SET) {
        if (param == "1") {
            if (fs_init()) {
                frames.push_back(frame_build(OperationType::RES, STORAGE_GROUP, MOUNT_COMMAND, "SD_MOUNT_OK"));
                return frames;
            } else {
                error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET);
                frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND, error_msg));
                return frames;
            }
        } else if (param == "0") {
            if (fs_unmount("/") == 0) {
                sd_card_mounted = false;
                frames.push_back(frame_build(OperationType::RES, STORAGE_GROUP, MOUNT_COMMAND, "SD_UNMOUNT_OK"));
                return frames;
            } else {
                error_msg = error_code_to_string(ErrorCode::FAIL_TO_SET);
                frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND, error_msg));
                return frames;
            }
        } else {
            error_msg = error_code_to_string(ErrorCode::PARAM_INVALID);
            frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND, error_msg));
            return frames;
        }
    } else {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, STORAGE_GROUP, MOUNT_COMMAND, error_msg));
        return frames;
    }
} // StorageCommands
```

8.27 lib/comms/commands/telemetry_commands.cpp File Reference

```
#include "commands.h"
#include "communication.h"
#include "telemetry_manager.h"
```

Include dependency graph for telemetry_commands.cpp:

Macros

- #define TELEMETRY_GROUP 8
- #define GET_LAST_TELEMETRY_RECORD_COMMAND 2
- #define GET_LAST_SENSOR_RECORD_COMMAND 3

Functions

- std::vector< Frame > handle_get_last_telemetry_record (const std::string &param, OperationType operationType)

  Handles the get last record command.
- std::vector< Frame > handle_get_last_sensor_record (const std::string &param, OperationType operation↩ Type)

  Handles the get last sensor record command.

Variables

- mutex_t telemetry_mutex

  Mutex for thread-safe access to the telemetry buffer.
- size_t telemetry_buffer_count
- size_t telemetry_buffer_write_index
- const int TELEMETRY_BUFFER_SIZE

  Circular buffer for telemetry records.

8.27.1 Macro Definition Documentation

8.27.1.1 TELEMETRY_GROUP

```
#define TELEMETRY_GROUP 8
```

Definition at line 5 of file telemetry_commands.cpp.

8.27.1.2 GET_LAST_TELEMETRY_RECORD_COMMAND

```
#define GET_LAST_TELEMETRY_RECORD_COMMAND 2
```

Definition at line 6 of file telemetry_commands.cpp.

8.27.1.3 GET_LAST_SENSOR_RECORD_COMMAND

```
#define GET_LAST_SENSOR_RECORD_COMMAND 3
```

Definition at line 7 of file telemetry_commands.cpp.

8.27.2 Variable Documentation

8.27.2.1 telemetry_mutex

```
mutex_t telemetry_mutex  [extern]
```

Mutex for thread-safe access to the telemetry buffer.

Definition at line 74 of file telemetry_manager.cpp.

8.27.2.2 telemetry_buffer_count

```
size_t telemetry_buffer_count  [extern]
```

Definition at line 61 of file telemetry_manager.cpp.

8.27.2.3 telemetry_buffer_write_index

```
size_t telemetry_buffer_write_index  [extern]
```

Definition at line 62 of file telemetry_manager.cpp.

8.28 telemetry_commands.cpp

Go to the documentation of this file.

```
#include "commands.h"
#include "communication.h"
#include "telemetry_manager.h"

#define TELEMETRY_GROUP 8
#define GET_LAST_TELEMETRY_RECORD_COMMAND 2
#define GET_LAST_SENSOR_RECORD_COMMAND 3

extern mutex_t telemetry_mutex;
extern size_t telemetry_buffer_count;
extern size_t telemetry_buffer_write_index;
extern const int TELEMETRY_BUFFER_SIZE;


std::vector<Frame> handle_get_last_telemetry_record(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, TELEMETRY_GROUP, GET_LAST_TELEMETRY_RECORD_COMMAND, error_msg));
        return frames;
    }

    std::string csv_data = get_last_telemetry_record_csv();

    if (csv_data.empty()) {
        error_msg = "NO_DATA";
        frames.push_back(frame_build(OperationType::ERR, TELEMETRY_GROUP, GET_LAST_SENSOR_RECORD_COMMAND, error_msg));
        return frames;
    }
    // Create and send the frame with the base64 encoded data
    frames.push_back(frame_build(OperationType::VAL, TELEMETRY_GROUP, GET_LAST_TELEMETRY_RECORD_COMMAND, csv_data));

    return frames;
}


std::vector<Frame> handle_get_last_sensor_record(const std::string& param, OperationType operationType) {
    std::vector<Frame> frames;
    std::string error_msg;

    if (operationType != OperationType::GET) {
        error_msg = error_code_to_string(ErrorCode::INVALID_OPERATION);
        frames.push_back(frame_build(OperationType::ERR, TELEMETRY_GROUP, GET_LAST_SENSOR_RECORD_COMMAND, error_msg));
        return frames;
    }

    std::string csv_data = get_last_sensor_record_csv();

    if (csv_data.empty()) {
```

```
        error_msg = "NO_DATA";
        frames.push_back(frame_build(OperationType::ERR, TELEMETRY_GROUP, GET_LAST_SENSOR_RECORD_COMMAND, error_msg));
        return frames;
    }

    // Create and send the frame with the sensor data
    frames.push_back(frame_build(OperationType::VAL, TELEMETRY_GROUP, GET_LAST_SENSOR_RECORD_COMMAND, csv_data));

    return frames;
} // TelemetryBufferCommands
```

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.
- void lora_tx_done_callback ()

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 18 of file communication.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.29.1.2 lora_tx_done_callback()

```
void lora_tx_done_callback ()
```

Definition at line 43 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.

```
#include "communication.h"

string outgoing;
uint8_t msgCount = 0;
long lastSendTime = 0;
long lastReceiveTime = 0;
long lastPrintTime = 0;
unsigned long interval = 0;


// In initialize_radio() function in communication.cpp
bool initialize_radio() {
    LoRa.set_pins(lora_cs_pin, lora_reset_pin, lora_irq_pin);
    long frequency = 433E6;
    bool init_status = false;
    if (!LoRa.begin(frequency))
    {
        uart_print("LoRa init failed. Check your connections.", VerbosityLevel::WARNING);
        init_status = false;
    } else {
        uart_print("LoRa initialized with frequency " + std::to_string(frequency), VerbosityLevel::INFO);

        // Set up TxDone callback to automatically return to receive mode
        LoRa.onTxDone(lora_tx_done_callback);

        // Start in receive mode
        LoRa.receive(0);

        init_status = true;
    }

    EventEmitter::emit(EventGroup::COMMS, init_status ? CommsEvent::RADIO_INIT : CommsEvent::RADIO_ERROR);

    return init_status;
}

void lora_tx_done_callback() {
    uart_print("LoRa transmission complete", VerbosityLevel::DEBUG);
    LoRa.receive(0);
}
```

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 lora_tx_done_callback ()
- void on_receive (int packetSize)

  Callback function for handling received LoRa packets.
- void handle_uart_input ()

  Handles UART input.
- void send_message (std::string outgoing)
- void send_frame_uart (const Frame &frame)
- void send_frame_lora (const Frame &frame)
- void split_and_send_message (const uint8_t *data, size_t length)

  Sends a large packet using LoRa.
- std::vector< Frame > execute_command (uint32_t commandKey, const std::string &param, OperationType operationType)

  Executes a command based on its key.
- void frame_process (const std::string &data, Interface interface)

  Executes a command based on the command key and the parameter.
- std::string frame_encode (const Frame &frame)

  Encodes a Frame instance into a string.
- Frame frame_decode (const std::string &data)

  Decodes a string into a Frame instance.
- Frame frame_build (OperationType operation, uint8_t group, uint8_t command, const std::string &value, const ValueUnit unitType=ValueUnit::UNDEFINED)

  Builds a Frame instance based on the execution result, group, command, value, and unit.
- std::string determine_unit (uint8_t group, uint8_t command)

8.31.1 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 18 of file communication.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.2 lora_tx_done_callback()

```
void lora_tx_done_callback ()
```

Definition at line 43 of file communication.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.3 on_receive()

```
void on_receive (
            int packet_size)
```

Callback function for handling received LoRa packets.

Parameters

packet_size	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:

8.31.1.4 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 79 of file receive.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.5 send_message()

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

8.31.1.6 send_frame_uart()

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

Definition at line 47 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 40 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

data	The data to send.
length	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 lora_tx_done_callback();
```

```
void on_receive(int packetSize);
void handle_uart_input();
void send_message(std::string outgoing);
void send_frame_uart(const Frame& frame);
void send_frame_lora(const Frame& frame);

void split_and_send_message(const uint8_t* data, size_t length);

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

void frame_process(const std::string& data, Interface interface);
std::string frame_encode(const Frame& frame);
Frame frame_decode(const std::string& data);
Frame frame_build(OperationType operation, uint8_t group, uint8_t command,const std::string& value, const ValueUnit unitType  = ValueUnit::UNDEFINED);

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

#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 > command_handlers

  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> command_handlers;
00005 extern volatile uint16_t eventRegister;
00006
00014
00037 std::string frame_encode(const Frame& frame) {
00038     std::stringstream ss;
00039     ss « static_cast<int>(frame.direction) « DELIMITER
00040        « operation_type_to_string(frame.operationType) « DELIMITER
00041        « static_cast<int>(frame.group) « DELIMITER
00042        « static_cast<int>(frame.command) « DELIMITER
00043        « frame.value;
00044
00045     if (!frame.unit.empty()) {
00046         ss « DELIMITER « frame.unit;
00047     }
00048
00049     return FRAME_BEGIN + DELIMITER + ss.str() + DELIMITER + FRAME_END;
00050 }
00051
00052
00062 Frame frame_decode(const std::string& data) {
00063     try {
00064         Frame frame;
00065         std::stringstream ss(data);
00066         std::string token;
00067
00068         std::getline(ss, token, DELIMITER);
00069         if (token != FRAME_BEGIN)
00070             throw std::runtime_error("Invalid frame header");
00071         frame.header = token;
00072
00073         std::string decoded_frame_data;
00074         while (std::getline(ss, token, DELIMITER)) {
00075             if (token == FRAME_END) break;
00076             decoded_frame_data += token + DELIMITER;
00077         }
00078         if (!decoded_frame_data.empty()) {
00079             decoded_frame_data.pop_back();
```

```
        }

        std::stringstream frame_data_stream(decoded_frame_data);

        std::getline(frame_data_stream, token, DELIMITER);
        frame.direction = std::stoi(token);

        std::getline(frame_data_stream, token, DELIMITER);
        frame.operationType = string_to_operation_type(token);

        std::getline(frame_data_stream, token, DELIMITER);
        frame.group = std::stoi(token);

        std::getline(frame_data_stream, token, DELIMITER);
        frame.command = std::stoi(token);

        std::getline(frame_data_stream, token, DELIMITER);
        frame.value = token;

        std::getline(frame_data_stream, token, DELIMITER);
        frame.unit = token;

        return frame;
    } catch (const std::exception& e) {
        uart_print("Frame error: " + std::string(e.what()), VerbosityLevel::ERROR);
        Frame error_frame = frame_build(OperationType::ERR, 0, 0, e.what());
        return error_frame;
    }
}


void frame_process(const std::string& data, Interface interface) {
    gpio_put(PICO_DEFAULT_LED_PIN, 0);

    uart_print("Processing frame: " + data, VerbosityLevel::WARNING);
    try {
        Frame frame = frame_decode(data);
        uint32_t command_key = (static_cast<uint32_t>(frame.group) << 8) |
static_cast<uint32_t>(frame.command);

        std::vector<Frame> response_frames = execute_command(command_key, frame.value,
frame.operationType);

        gpio_put(PICO_DEFAULT_LED_PIN, 1);

        // Send all responses through the same interface that received the command
        for (const auto& response_frame : response_frames) {
            if (interface == Interface::UART) {
                send_frame_uart(response_frame);
            } else if (interface == Interface::LORA) {
                send_frame_lora(response_frame);
                sleep_ms(25);
            }
        }
    } catch (const std::exception& e) {
        Frame error_frame = frame_build(OperationType::ERR, 0, 0, e.what());
        if (interface == Interface::UART) {
            send_frame_uart(error_frame);
        } else if (interface == Interface::LORA) {
            send_frame_lora(error_frame);
        }
    }
}

Frame frame_build(OperationType operation, uint8_t group, uint8_t command,
                const std::string& value, const ValueUnit unitType) {
    Frame frame;
    frame.header = FRAME_BEGIN;
    frame.footer = FRAME_END;

    switch (operation) {
        case OperationType::VAL:
            frame.direction = 1;
            frame.operationType = OperationType::VAL;
            frame.value = value;
            frame.unit = value_unit_type_to_string(unitType);
            break;

        case OperationType::ERR:
            frame.direction = 1;
            frame.operationType = OperationType::ERR;
            frame.value = value;
            frame.unit = value_unit_type_to_string(ValueUnit::UNDEFINED);
            break;

        case OperationType::RES:
            frame.direction = 1;
```

```
            frame.operationType = OperationType::RES;
            frame.value = value;
            frame.unit = value_unit_type_to_string(unitType);
            break;

        case OperationType::SEQ:
            frame.direction = 1;
            frame.operationType = OperationType::SEQ;
            frame.value = value;
            frame.unit = value_unit_type_to_string(unitType);
            break;
    }

    frame.group = group;
    frame.command = command;

    return frame;
}
 // 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 ErrorCode {
  PARAM_UNNECESSARY , PARAM_REQUIRED , PARAM_INVALID , INVALID_OPERATION ,
  NOT_ALLOWED , INVALID_FORMAT , INVALID_VALUE , FAIL_TO_SET ,
  INTERNAL_FAIL_TO_READ , UNKNOWN_ERROR }

  Standard error codes for command responses.
- enum class OperationType {
  GET , SET , RES , VAL ,
  SEQ , ERR }

  Represents the type of operation being performed.
- enum class CommandAccessLevel { NONE , READ_ONLY , WRITE_ONLY , READ_WRITE }

  Represents the access level required to execute a command.
- enum class ValueUnit {
  UNDEFINED , SECOND , VOLT , BOOL ,
  DATETIME , TEXT , MILIAMP }

  Represents the unit of measurement for a payload value.
- enum class ExceptionType {
  NONE , NOT_ALLOWED , INVALID_PARAM , INVALID_OPERATION ,
  PARAM_UNECESSARY }

  Represents the type of exception that occurred during command execution.
- enum class Interface { UART , LORA }

  Represents the communication interface being used.

Functions

- std::string exception_type_to_string (ExceptionType type)

  Converts an ExceptionType to a string.
- std::string error_code_to_string (ErrorCode code)

  Converts an ErrorCode to its string representation.
- std::string operation_type_to_string (OperationType type)

  Converts an OperationType to a string.
- OperationType string_to_operation_type (const std::string &str)

  Converts a string to an OperationType.
- std::vector< uint8_t > hex_string_to_bytes (const std::string &hexString)

  Converts a hex string to a vector of bytes.
- std::string value_unit_type_to_string (ValueUnit unit)

  Converts a ValueUnit to a string.

Variables

- const std::string FRAME_BEGIN = "KBST"
- const std::string FRAME_END = "TSBK"
- const char DELIMITER = ';'

8.35.1 Enumeration Type Documentation

8.35.1.1 ErrorCode

```
enum class ErrorCode  [strong]
```

Standard error codes for command responses.

Enumerator

PARAM_UNNECESSARY	
PARAM_REQUIRED	
PARAM_INVALID	
INVALID_OPERATION	
NOT_ALLOWED	
INVALID_FORMAT	
INVALID_VALUE	
FAIL_TO_SET	
INTERNAL_FAIL_TO_READ	
UNKNOWN_ERROR	

Definition at line 45 of file protocol.h.

8.35.1.2 OperationType

```
enum class OperationType [strong]
```

Represents the type of operation being performed.

Enumerator

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

Definition at line 63 of file protocol.h.

8.35.1.3 CommandAccessLevel

```
enum class CommandAccessLevel [strong]
```

Represents the access level required to execute a command.

Enumerator

NONE	No access allowed.
READ_ONLY	Read-only access.
WRITE_ONLY	Write-only access.
READ_WRITE	Read and write access.

Definition at line 85 of file protocol.h.

8.35.1.4 ValueUnit

```
enum class ValueUnit [strong]
```

Represents the unit of measurement for a payload value.

Enumerator

UNDEFINED	Unit is undefined.
SECOND	Unit is seconds.
VOLT	Unit is volts.
BOOL	Unit is boolean.
DATETIME	Unit is date and time.
TEXT	Unit is text.
MILIAMP	Unit is milliamperes.

Definition at line 102 of file protocol.h.

8.35.1.5 ExceptionType

```
enum class ExceptionType [strong]
```

Represents the type of exception that occurred during command execution.

Enumerator

NONE	No exception.
NOT_ALLOWED	Operation not allowed.
INVALID_PARAM	Invalid parameter provided.
INVALID_OPERATION	Invalid operation requested.
PARAM_UNECESSARY	Parameter is unnecessary for the operation.

Definition at line 125 of file protocol.h.

8.35.1.6 Interface

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

Represents the communication interface being used.

Enumerator

UART	UART interface.
LORA	LoRa interface.

Definition at line 144 of file protocol.h.

8.35.2 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

type	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 error_code_to_string()

```
std::string error_code_to_string (
            ErrorCode code)
```

Converts an ErrorCode to its string representation.

Parameters

code	The error code

Returns

String representation of the error code

Definition at line 83 of file utils_converters.cpp.

Here is the caller graph for this function:

8.35.2.3 operation_type_to_string()

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

Converts an OperationType to a string.

Parameters

type	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:

8.35.2.4 string_to_operation_type()

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

Converts a string to an OperationType.

Parameters

str	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:

8.35.2.5 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

hexString	The hex string to convert.

Returns

A vector of bytes representing the hex string.

Definition at line 104 of file utils_converters.cpp.

Here is the caller graph for this function:

8.35.2.6 value_unit_type_to_string()

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

Converts a ValueUnit to a string.

Parameters

unit	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 ErrorCode {
00046     PARAM_UNNECESSARY,     // Parameter provided but not needed
00047     PARAM_REQUIRED,        // Required parameter missing
00048     PARAM_INVALID,         // Parameter has invalid format or value
00049     INVALID_OPERATION,     // Operation not allowed for this command
00050     NOT_ALLOWED,           // Operation not permitted
00051     INVALID_FORMAT,        // Input format is incorrect
00052     INVALID_VALUE,         // Value is outside expected range
00053     FAIL_TO_SET,           // Failed to set provided value
00054     INTERNAL_FAIL_TO_READ,// Failed to read from device in remote
00055     UNKNOWN_ERROR          // Generic error
00056 };
00057
00058
```

```
enum class OperationType {
    GET,
    SET,
    RES,
    VAL,
    SEQ,
    ERR,

};



enum class CommandAccessLevel {
    NONE,
    READ_ONLY,
    WRITE_ONLY,
    READ_WRITE
};



enum class ValueUnit {
    UNDEFINED,
    SECOND,
    VOLT,
    BOOL,
    DATETIME,
    TEXT,
    MILIAMP,
};



enum class ExceptionType {
    NONE,
    NOT_ALLOWED,
    INVALID_PARAM,
    INVALID_OPERATION,
    PARAM_UNECESSARY
};



enum class Interface {
    UART,
    LORA
};


struct Frame {
    std::string header;             // Start marker
    uint8_t direction;              // 0 = ground->sat, 1 = sat->ground
    OperationType operationType;
    uint8_t group;                  // Group ID
    uint8_t command;                // Command ID within group
    std::string value;              // Payload value
    std::string unit;               // Payload unit
    std::string footer;             // End marker
};

std::string exception_type_to_string(ExceptionType type);
std::string error_code_to_string(ErrorCode code);
std::string operation_type_to_string(OperationType type);
OperationType string_to_operation_type(const std::string& str);
std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString);
std::string value_unit_type_to_string(ValueUnit unit);

#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 packet_size)

  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 packet_size)
```

Callback function for handling received LoRa packets.

Parameters

packet_size	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 79 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.

```
#include "communication.h"



void on_receive(int packet_size) {
    if (packet_size == 0) return;
    uart_print("Received LoRa packet of size " + std::to_string(packet_size), VerbosityLevel::DEBUG);

    uint8_t buffer[256];
    int bytes_read = 0;

    while (LoRa.available() && bytes_read < packet_size) {
        buffer[bytes_read++] = LoRa.read();
    }

    uart_print("Received " + std::to_string(bytes_read) + " bytes", VerbosityLevel::DEBUG);

    // Extract LoRa metadata
    uint8_t received_destination = buffer[0];
    uint8_t received_local_address = buffer[1];

    // Validate metadata (optional, for security)
    if (received_destination != lora_address_local) {
        uart_print("Error: Destination address mismatch!", VerbosityLevel::ERROR);
        return;
    }

    if (received_local_address != lora_address_remote) {
        uart_print("Error: Local address mismatch!", VerbosityLevel::ERROR);
        return;
    }

    // Find the starting index of the actual frame data
    int start_index = 2; // Start after the metadata

    // Extract the frame data
    std::string received = std::string(reinterpret_cast<char*>(buffer + start_index), bytes_read - start_index);

    if (received.empty()) return;

    // Debug: Print raw hex values
    std::stringstream hex_dump;
    hex_dump « "Raw bytes: ";
    for (int i = 0; i < bytes_read; i++) {
        hex_dump « std::hex « std::setfill('0') « std::setw(2)
                « static_cast<int>(buffer[i]) « " ";
    }
    uart_print(hex_dump.str(), VerbosityLevel::DEBUG);

    // Find frame boundaries
    size_t header_pos = received.find(FRAME_BEGIN);
    size_t footer_pos = received.find(FRAME_END);

    if (header_pos != std::string::npos && footer_pos != std::string::npos && footer_pos > header_pos) {
        std::string frame_data = received.substr(header_pos, footer_pos + FRAME_END.length() - header_pos);
        uart_print("Extracted frame (length=" + std::to_string(frame_data.length()) + "): " + frame_data, VerbosityLevel::DEBUG);
        frame_process(frame_data, Interface::LORA);
    } else {
        uart_print("No valid frame found in received data", VerbosityLevel::WARNING);
    }
}


void handle_uart_input() {
    static std::string uart_buffer;

    while (uart_is_readable(DEBUG_UART_PORT)) {
        char c = uart_getc(DEBUG_UART_PORT);

        if (c == '\r' || c == '\n') {
            uart_print("Received UART string: " + uart_buffer, VerbosityLevel::DEBUG);
            frame_process(uart_buffer, Interface::UART);
            uart_buffer.clear();
        } else {
            uart_buffer += c;
        }
    }
}
```

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 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

outgoing	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 40 of file send.cpp.

8.39.2.3 send_frame_uart()

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

Definition at line 47 of file send.cpp.

8.39.2.4 split_and_send_message()

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

Sends a large packet using LoRa.

Parameters

data	The data to send.
length	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.

```
#include "communication.h"



void send_message(string outgoing)
{
    int n = outgoing.length();
    char send[n + 1];
    strcpy(send, outgoing.c_str());

    uart_print("LoRa packet begin", VerbosityLevel::DEBUG);
    LoRa.beginPacket();        // start packet
    LoRa.write(lora_address_remote);  // add destination address
    LoRa.write(lora_address_local); // add sender address
    LoRa.print(send);          // add payload
    LoRa.endPacket(false);     // finish packet and send it

    uart_print("LoRa packet end", VerbosityLevel::DEBUG);

    std::string message_to_log = "Sent message of size " + std::to_string(n);
    message_to_log += " to 0x" + std::to_string(lora_address_remote);
    message_to_log += " containing: " + string(send);

    uart_print(message_to_log, VerbosityLevel::DEBUG);

    LoRa.flush();
}


void send_frame_lora(const Frame& frame) {
    uart_print("Sending frame via LoRa", VerbosityLevel::DEBUG);
    string outgoing = frame_encode(frame);
    send_message(outgoing);
    uart_print("Frame sent via LoRa", VerbosityLevel::DEBUG);
}

void send_frame_uart(const Frame& frame) {
    std::string encoded_frame = frame_encode(frame);
    uart_print(encoded_frame);
}


void split_and_send_message(const uint8_t* data, size_t length)
{
    const size_t MAX_PKT_SIZE = 255;
    size_t offset = 0;
    while (offset < length)
    {
        size_t chunk_size = ((length - offset) < MAX_PKT_SIZE) ? (length - offset) : MAX_PKT_SIZE;
        LoRa.beginPacket();
        LoRa.write(&data[offset], chunk_size);
        LoRa.endPacket();
        offset += chunk_size;
        sleep_ms(100);
    }
}

```

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::string error_code_to_string (ErrorCode code)

  Converts an ErrorCode to its string representation.
- std::vector< uint8_t > hex_string_to_bytes (const std::string &hexString)

  Converts a hex string to a vector of bytes.

8.41.1 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

type	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

unit	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

type	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

str	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 error_code_to_string()

```
std::string error_code_to_string (
            ErrorCode code)
```

Converts an ErrorCode to its string representation.

Parameters

code	The error code

Returns

String representation of the error code

Definition at line 83 of file utils_converters.cpp.

Here is the caller graph for this function:

8.41.2.6 hex_string_to_bytes()

```
std::vector< uint8_t > hex_string_to_bytes (
            const std::string & hexString)
```

Converts a hex string to a vector of bytes.

Parameters

hexString	The hex string to convert.

Returns

A vector of bytes representing the hex string.

Definition at line 104 of file utils_converters.cpp.

Here is the caller graph for this function:

8.42 utils_converters.cpp

Go to the documentation of this file.

```
#include "communication.h"


std::string exception_type_to_string(ExceptionType type) {
    switch (type) {
        case ExceptionType::NOT_ALLOWED:       return "NOT ALLOWED";
        case ExceptionType::INVALID_PARAM:     return "INVALID PARAM";
        case ExceptionType::INVALID_OPERATION: return "INVALID OPERATION";
        case ExceptionType::PARAM_UNECESSARY:  return "PARAM UNECESSARY";
        case ExceptionType::NONE:              return "NONE";
        default:                               return "UNKNOWN EXCEPTION";
    }
}


std::string value_unit_type_to_string(ValueUnit unit) {
    switch (unit) {
        case ValueUnit::UNDEFINED:  return "";
        case ValueUnit::SECOND:     return "s";
        case ValueUnit::VOLT:       return "V";
        case ValueUnit::BOOL:       return "";
        case ValueUnit::DATETIME:   return "";
        case ValueUnit::TEXT:       return "";
        case ValueUnit::MILIAMP:    return "mA";
        default: return "";
    }
}


std::string operation_type_to_string(OperationType type) {
    switch (type) {
        case OperationType::GET: return "GET";
        case OperationType::SET: return "SET";
        case OperationType::VAL: return "VAL";
        case OperationType::ERR: return "ERR";
        case OperationType::RES: return "RES";
        case OperationType::SEQ: return "SEQ";
        default: return "UNKNOWN";
    }
}


OperationType string_to_operation_type(const std::string& str) {
    if (str == "GET") return OperationType::GET;
```

```
    if (str == "SET") return OperationType::SET;
    if (str == "VAL") return OperationType::VAL;
    if (str == "ERR") return OperationType::ERR;
    if (str == "RES") return OperationType::RES;
    if (str == "SEQ") return OperationType::SEQ;
    return OperationType::GET; // Default to GET
}

std::string error_code_to_string(ErrorCode code) {
    switch (code) {
        case ErrorCode::PARAM_UNNECESSARY:      return "PARAM_UNNECESSARY";
        case ErrorCode::PARAM_REQUIRED:         return "PARAM_REQUIRED";
        case ErrorCode::PARAM_INVALID:          return "PARAM_INVALID";
        case ErrorCode::INVALID_OPERATION:      return "INVALID_OPERATION";
        case ErrorCode::NOT_ALLOWED:            return "NOT_ALLOWED";
        case ErrorCode::INVALID_FORMAT:         return "INVALID_FORMAT";
        case ErrorCode::INVALID_VALUE:          return "INVALID_VALUE";
        case ErrorCode::FAIL_TO_SET:            return "FAIL_TO_SET";
        case ErrorCode::INTERNAL_FAIL_TO_READ:  return "INTERNAL_FAIL_TO_READ";
        default:                                return "UNKNOWN_ERROR";
    }
}


std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString) {
    std::vector<uint8_t> bytes;
    for (size_t i = 0; i < hexString.length(); i += 2) {
        std::string byteString = hexString.substr(i, 2);
        unsigned int byte;
        std::stringstream ss;
        ss « std::hex « byteString;
        ss » byte;
        bytes.push_back(static_cast<uint8_t>(byte));
    }
    return bytes;
}
```

8.43 lib/eventman/event_manager.cpp File Reference

Implements the event management system for the Kubisat 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:

Variables

- volatile uint16_t eventLogId = 0

  Global event log ID counter.
- DS3231 systemClock

  External declaration of the system clock.
- EventManagerImpl eventManager

  Global instance of the EventManager implementation.

8.43.1 Detailed Description

Implements the event management system for the Kubisat 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.

```
#include "event_manager.h"
#include <cstdio>
#include "protocol.h"
#include "pico/multicore.h"
#include "communication.h"
#include "utils.h"
#include "DS3231.h"


volatile uint16_t eventLogId = 0;

extern DS3231 systemClock;

EventManagerImpl eventManager;

uint16_t EventManager::nextEventId = 0;

void EventManager::log_event(uint8_t group, uint8_t event) {
    mutex_enter_blocking(&eventMutex);

    // Clear buffer if it's full
    if (eventCount >= EVENT_BUFFER_SIZE) {
        eventCount = 0;
        writeIndex = 0;
    }

    EventLog& log = events[writeIndex];
    log.id = nextEventId++;
    log.timestamp = systemClock.get_unix_time();
    log.group = group;
    log.event = event;

    // Print event immediately
    uart_print(log.to_string(), VerbosityLevel::EVENT);

    writeIndex = (writeIndex + 1) % EVENT_BUFFER_SIZE;
    eventCount++;
    eventsSinceFlush++;

    // Flush to storage every EVENT_FLUSH_THRESHOLD events
    if (eventsSinceFlush >= EVENT_FLUSH_THRESHOLD) {
        save_to_storage();
        eventsSinceFlush = 0;
    }

    mutex_exit(&eventMutex);
}


const EventLog& EventManager::get_event(size_t index) const {
    static const EventLog emptyEvent = {0, 0, 0, 0};  // Initialize {id, timestamp, group, event}
    if (index >= eventCount) {
        return emptyEvent;
    }

    // Calculate actual index in circular buffer
    size_t actualIndex;
    if (eventCount == EVENT_BUFFER_SIZE) {
        actualIndex = (writeIndex + index) % EVENT_BUFFER_SIZE;
    } else {
        actualIndex = index;
    }

    return events[actualIndex];
}
```

8.45 lib/eventman/event_manager.h File Reference

Manages the event logging system for the Kubisat firmware.

```
#include "PowerManager.h"
#include <cstdint>
#include <string>
#include "pico/mutex.h"
#include "storage.h"
#include "utils.h"
```

Include dependency graph for event_manager.h:

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

Classes

- class EventLog

  Represents a single event log entry.
- class EventManager

  Manages the event logging system.
- class EventManagerImpl

  Implementation of the EventManager class.
- class EventEmitter

  Provides a static method for emitting events.

Macros

- #define EVENT_BUFFER_SIZE 100
- #define EVENT_FLUSH_THRESHOLD 10
- #define EVENT_LOG_FILE "/event_log.csv"

Enumerations

- enum class EventGroup : uint8_t { EventGroup::SYSTEM = 0x00 , EventGroup::POWER = 0x01 , EventGroup::COMMS = 0x02 , EventGroup::GPS = 0x03 , EventGroup::CLOCK = 0x04 }

  Represents the group to which an event belongs.
- enum class SystemEvent : uint8_t { SystemEvent::BOOT = 0x01 , SystemEvent::SHUTDOWN = 0x02 , SystemEvent::WATCHDOG_RESET = 0x03 , SystemEvent::CORE1_START = 0x04 , SystemEvent::CORE1_STOP = 0x05 }

  Represents specific system events.
- enum class PowerEvent : uint8_t { PowerEvent::LOW_BATTERY = 0x01 , PowerEvent::OVERCHARGE = 0x02 , PowerEvent::POWER_FALLING = 0x03 , PowerEvent::POWER_NORMAL = 0x04 , PowerEvent::SOLAR_ACTIVE = 0x05 , PowerEvent::SOLAR_INACTIVE = 0x06 , PowerEvent::USB_CONNECTED = 0x07 , PowerEvent::USB_DISCONNECTED = 0x08 }

  Represents specific power-related events.
- enum class CommsEvent : uint8_t { CommsEvent::RADIO_INIT = 0x01 , CommsEvent::RADIO_ERROR = 0x02 , CommsEvent::MSG_RECEIVED = 0x03 , CommsEvent::MSG_SENT = 0x04 , CommsEvent::UART_ERROR = 0x06 }

  Represents specific communication-related events.
- enum class GPSEvent : uint8_t { GPSEvent::LOCK = 0x01 , GPSEvent::LOST = 0x02 , GPSEvent::ERROR = 0x03 , GPSEvent::POWER_ON = 0x04 , GPSEvent::POWER_OFF = 0x05 , GPSEvent::DATA_READY = 0x06 , GPSEvent::PASS_THROUGH_START = 0x07 , GPSEvent::PASS_THROUGH_END = 0x08 }

  Represents specific GPS-related events.
- enum class ClockEvent : uint8_t { ClockEvent::CHANGED = 0x01 , ClockEvent::GPS_SYNC = 0x02 , ClockEvent::GPS_SYNC_DATA_NOT_READY = 0x03 }

  Represents specific clock-related events.

Functions

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

  Converts the EventLog to a string representation.

Variables

- uint16_t id

  Sequence number.
- uint32_t timestamp

  Unix timestamp or system time.
- uint8_t group

  Event group identifier.
- uint8_t event

  Specific event identifier.
- class EventManager __attribute__
- EventManagerImpl eventManager

  Global instance of the EventManagerImpl class.

8.45.1 Detailed Description

Manages the event logging system for the Kubisat firmware.

Definition in file event_manager.h.

8.45.2 Macro Definition Documentation

8.45.2.1 EVENT_BUFFER_SIZE

```
#define EVENT_BUFFER_SIZE 100
```

Definition at line 11 of file event_manager.h.

8.45.2.2 EVENT_FLUSH_THRESHOLD

```
#define EVENT_FLUSH_THRESHOLD 10
```

Definition at line 12 of file event_manager.h.

8.45.2.3 EVENT_LOG_FILE

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

Definition at line 13 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"
00010
00011 #define EVENT_BUFFER_SIZE 100
00012 #define EVENT_FLUSH_THRESHOLD 10
00013 #define EVENT_LOG_FILE "/event_log.csv"
00014
00022
00023
00028 enum class EventGroup : uint8_t {
00030     SYSTEM  = 0x00,
00032     POWER   = 0x01,
00034     COMMS   = 0x02,
00036     GPS     = 0x03,
00038     CLOCK   = 0x04
00039 };
00040
00045 enum class SystemEvent : uint8_t {
00047     BOOT           = 0x01,
00049     SHUTDOWN       = 0x02,
00051     WATCHDOG_RESET = 0x03,
00053     CORE1_START    = 0x04,
00055     CORE1_STOP     = 0x05
00056 };
00057
00062 enum class PowerEvent : uint8_t {
00064     LOW_BATTERY      = 0x01,
00066     OVERCHARGE       = 0x02,
00068     POWER_FALLING    = 0x03,
00070     POWER_NORMAL     = 0x04,
```

```
    SOLAR_ACTIVE     = 0x05,
    SOLAR_INACTIVE   = 0x06,
    USB_CONNECTED    = 0x07,
    USB_DISCONNECTED = 0x08
};

enum class CommsEvent : uint8_t {
    RADIO_INIT    = 0x01,
    RADIO_ERROR   = 0x02,
    MSG_RECEIVED  = 0x03,
    MSG_SENT      = 0x04,
    UART_ERROR    = 0x06
};

enum class GPSEvent : uint8_t {
    LOCK                = 0x01,
    LOST                = 0x02,
    ERROR               = 0x03,
    POWER_ON            = 0x04,
    POWER_OFF           = 0x05,
    DATA_READY          = 0x06,
    PASS_THROUGH_START  = 0x07,
    PASS_THROUGH_END    = 0x08
};


enum class ClockEvent : uint8_t {
    CHANGED  = 0x01,
    GPS_SYNC = 0x02,
    GPS_SYNC_DATA_NOT_READY = 0x03
};


class EventLog {
    public:
        uint16_t id;
        uint32_t timestamp;
        uint8_t group;
        uint8_t event;

        std::string to_string() const {
            char buffer[256];
            snprintf(buffer, sizeof(buffer),
                        "EventLog: id=%u, timestamp=%lu, group=%u, event=%u",
                        id, timestamp, group, event);
            return std::string(buffer);
        }
    } __attribute__((packed));


class EventManager {
    public:
        EventManager()
            : eventCount(0)
            , writeIndex(0)
            , eventsSinceFlush(0)  // Add eventsSinceFlush initialization
        {
            mutex_init(&eventMutex);
        }

        virtual ~EventManager() = default;

        virtual void init() {
            load_from_storage();
        }

        void log_event(uint8_t group, uint8_t event);

        const EventLog& get_event(size_t index) const;

        size_t get_event_count() const { return eventCount; }

        virtual bool save_to_storage() = 0;

        virtual bool load_from_storage() = 0;

        protected:
        EventLog events[EVENT_BUFFER_SIZE];
        size_t eventCount;
        size_t writeIndex;
        mutex_t eventMutex;
        static uint16_t nextEventId;
        size_t eventsSinceFlush;
    };


class EventManagerImpl : public EventManager {
```

```
    public:
        EventManagerImpl() {
            init(); // Safe to call virtual functions here
        }

        public:
        bool save_to_storage() override {
            if(!sd_card_mounted) {
                bool status = fs_init();
                if(!status) {
                    return false;
                }
            }

            FILE *file = fopen(EVENT_LOG_FILE, "a");
            if (file) {
                // Calculate start index for last EVENT_FLUSH_THRESHOLD events
                size_t startIdx = (writeIndex >= eventsSinceFlush) ?
                    writeIndex - eventsSinceFlush :
                    EVENT_BUFFER_SIZE - (eventsSinceFlush - writeIndex);

                // Write only the most recent batch of events
                for (size_t i = 0; i < eventsSinceFlush; i++) {
                    size_t idx = (startIdx + i) % EVENT_BUFFER_SIZE;
                    fprintf(file, "%u;%lu;%u;%u\n",
                        events[idx].id,
                        events[idx].timestamp,
                        events[idx].group,
                        events[idx].event
                    );
                }
                fclose(file);
                uart_print("Events saved to storage", VerbosityLevel::INFO);
                return true;
            }
            return false;
        }

        bool load_from_storage() override {
            // TODO: Implement based on chosen storage (SD/EEPROM)
            return false;
        }
    };


extern EventManagerImpl eventManager;

class EventEmitter {
    public:
    template<typename T>
    static void emit(EventGroup group, T event) {
        eventManager.log_event(
            static_cast<uint8_t>(group),
            static_cast<uint8_t>(event)
        );
    }
};

#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 256

Functions

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

Variables

- volatile bool pause_gps_collection
- NMEAData nmea_data

8.47.1 Macro Definition Documentation

8.47.1.1 MAX_RAW_DATA_LENGTH

```
#define MAX_RAW_DATA_LENGTH 256
```

Definition at line 15 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 19 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 29 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 pause_gps_collection

```
volatile bool pause_gps_collection  [extern]
```

Definition at line 8 of file main.cpp.

8.47.3.2 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.

```
// filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/gps_collector.cpp
#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>

extern volatile bool pause_gps_collection;

#define MAX_RAW_DATA_LENGTH 256

extern NMEAData nmea_data;

std::vector<std::string> splitString(const std::string& str, char delimiter) {
    std::vector<std::string> tokens;
    std::stringstream ss(str);
    std::string token;
    while (std::getline(ss, token, delimiter)) {
        tokens.push_back(token);
    }
    return tokens;
}

void collect_gps_data() {

    if (pause_gps_collection) {
        return;
    }

    static char raw_data_buffer[MAX_RAW_DATA_LENGTH];
    static int raw_data_index = 0;

    while (uart_is_readable(GPS_UART_PORT)) {
        char c = uart_getc(GPS_UART_PORT);

        if (c == '\r' || c == '\n') {
            // End of message
            if (raw_data_index > 0) {
                raw_data_buffer[raw_data_index] = '\0';
                std::string message(raw_data_buffer);
                raw_data_index = 0;

                // Split the message into tokens
                std::vector<std::string> tokens = splitString(message, ',');

                // Update the global vectors based on the sentence type
                if (message.find("$GPRMC") == 0) {
                    nmea_data.update_rmc_tokens(tokens);
                } else if (message.find("$GPGGA") == 0) {
                    nmea_data.update_gga_tokens(tokens);
                }
            }
        } else {
            // Append to buffer
            if (raw_data_index < MAX_RAW_DATA_LENGTH - 1) {
                raw_data_buffer[raw_data_index++] = c;
            } else {
                raw_data_index = 0;
            }
        }
    }
}
```

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 29 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.

```
#include "lib/location/NMEA/NMEA_data.h"

NMEAData nmea_data;

NMEAData::NMEAData() {
    mutex_init(&rmc_mutex_);
    mutex_init(&gga_mutex_);
}

void NMEAData::update_rmc_tokens(const std::vector<std::string>& tokens) {
    mutex_enter_blocking(&rmc_mutex_);
    rmc_tokens_ = tokens;
    mutex_exit(&rmc_mutex_);
}

void NMEAData::update_gga_tokens(const std::vector<std::string>& tokens) {
    mutex_enter_blocking(&gga_mutex_);
    gga_tokens_ = tokens;
    mutex_exit(&gga_mutex_);
}

std::vector<std::string> NMEAData::get_rmc_tokens() const {
    mutex_enter_blocking(const_cast<mutex_t*>(&rmc_mutex_));
    std::vector<std::string> copy = rmc_tokens_;
    mutex_exit(const_cast<mutex_t*>(&rmc_mutex_));
    return copy;
}

std::vector<std::string> NMEAData::get_gga_tokens() const {
    mutex_enter_blocking(const_cast<mutex_t*>(&gga_mutex_));
    std::vector<std::string> copy = gga_tokens_;
    mutex_exit(const_cast<mutex_t*>(&gga_mutex_));
    return copy;
}

bool NMEAData::has_valid_time() const {
    return rmc_tokens_.size() >= 10 && rmc_tokens_[1].length() > 5;
}

time_t NMEAData::get_unix_time() const {
    if (!has_valid_time()) {
        return 0; // Invalid time
    }

    // Parse date and time from RMC tokens
    // Format: hhmmss.sss,A,ddmm.mmmm,N,dddmm.mmmm,W,speed,course,ddmmyy
    std::string time_str = rmc_tokens_[1]; // hhmmss.sss
    std::string date_str = rmc_tokens_[9]; // ddmmyy

    if (time_str.length() < 6 || date_str.length() < 6) {
        return 0;
    }

    struct tm timeinfo = {0};

    // Parse time: hours (0-1), minutes (2-3), seconds (4-5)
    timeinfo.tm_hour = std::stoi(time_str.substr(0, 2));
```

```
    timeinfo.tm_min = std::stoi(time_str.substr(2, 2));
    timeinfo.tm_sec = std::stoi(time_str.substr(4, 2));

    // Parse date: day (0-1), month (2-3), year (4-5)
    timeinfo.tm_mday = std::stoi(date_str.substr(0, 2));
    timeinfo.tm_mon = std::stoi(date_str.substr(2, 2)) - 1; // Months from 0-11
    timeinfo.tm_year = std::stoi(date_str.substr(4, 2)) + 100; // Years since 1900

    // Convert to unix time
    return mktime(&timeinfo);
}
```

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

```
#include <vector>
#include <string>
#include "pico/sync.h"
#include <ctime>
```

Include dependency graph for NMEA_data.h:

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

Classes

- class NMEAData

Variables

- NMEAData nmea_data

8.53.1 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 #include <ctime>
00009
00010 class NMEAData {
00011 public:
00012     NMEAData();
00013     void update_rmc_tokens(const std::vector<std::string>& tokens);
00014     void update_gga_tokens(const std::vector<std::string>& tokens);
00015
00016     std::vector<std::string> get_rmc_tokens() const;
00017     std::vector<std::string> get_gga_tokens() const;
00018
00019     bool has_valid_time() const;
00020
00021     time_t get_unix_time() const;
00022
00023 private:
00024     std::vector<std::string> rmc_tokens_;
00025     std::vector<std::string> gga_tokens_;
00026     mutex_t rmc_mutex_;
00027     mutex_t gga_mutex_;
00028 };
00029
00030 extern NMEAData nmea_data;
00031
00032 #endif
```

8.55 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 spi1
```

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 spi0
```

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 spi0
```

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.

```
// pin_config.h
#include <stdint.h>

#ifndef PIN_CONFIG_H
#define PIN_CONFIG_H

//DEBUG uart
#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

// GPS configuration
#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  // NMEA sentences are usually under 85 chars

// SPI configuration for SD card
#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



// LoRa constants - declare as extern
extern const int lora_cs_pin;      // LoRa radio chip select
extern const int lora_reset_pin;   // LoRa radio reset
extern const int lora_irq_pin;     // LoRa hardware interrupt pin
extern uint8_t lora_address_local; // address of this device
extern uint8_t lora_address_remote;  // destination to send to


#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(ina3221_addr_t addr, i2c_inst_t* i2c)
00047     : _i2c_addr(addr), _i2c(i2c) {}
00048
00049
00056 bool INA3221::begin() {
00057     uart_print("INA3221 initializing...", VerbosityLevel::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);
```

```

    if (manuf_id == 0x5449 && die_id == 0x3220) {
       uart_print("INA3221 found and initialized.", VerbosityLevel::DEBUG);
        return true;
    } else {
        uart_print("INA3221 initialization failed. Incorrect IDs.", VerbosityLevel::ERROR);
        return false;
    }

}


void INA3221::reset(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.reset = 1;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


uint16_t INA3221::get_manufacturer_id() {
    uint16_t id = 0;
    _read(INA3221_REG_MANUF_ID, &id);
    return id;
}


uint16_t INA3221::get_die_id() {
    uint16_t id = 0;
    _read(INA3221_REG_DIE_ID, &id);
    return id;
}


uint16_t INA3221::read_register(ina3221_reg_t reg){
    uint16_t val = 0;
    _read(reg, &val);
    return val;
}


//configure

void INA3221::set_mode_power_down(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.mode_bus_en = 0;
    conf_reg.mode_continious_en =0 ;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_mode_continuous(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.mode_continious_en =1;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_mode_triggered(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.mode_continious_en = 0;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_shunt_measurement_enable(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.mode_shunt_en = 1;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_shunt_measurement_disable(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.mode_shunt_en = 0;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
```

```
}


void INA3221::set_bus_measurement_enable(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.mode_bus_en = 1;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_bus_measurement_disable(){
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.mode_bus_en = 0;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_averaging_mode(ina3221_avg_mode_t mode) {
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.avg_mode = mode;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_bus_conversion_time(ina3221_conv_time_t convTime) {
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.bus_conv_time = convTime;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


void INA3221::set_shunt_conversion_time(ina3221_conv_time_t convTime) {
    conf_reg_t conf_reg;

    _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
    conf_reg.shunt_conv_time = convTime;
    _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
}


//get measurement
int32_t INA3221::get_shunt_voltage(ina3221_ch_t channel) {
    int32_t res;
    ina3221_reg_t reg;
    uint16_t val_raw = 0;

    switch(channel){
        case INA3221_CH1:
            reg = INA3221_REG_CH1_SHUNTV;
            break;
        case INA3221_CH2:
            reg = INA3221_REG_CH2_SHUNTV;
            break;
        case INA3221_CH3:
            reg = INA3221_REG_CH3_SHUNTV;
            break;
    }

    _read(reg, &val_raw);

    res = (int16_t) (val_raw » 3);
    res *= SHUNT_VOLTAGE_LSB_UV;

    return res;
}


float INA3221::get_current_ma(ina3221_ch_t channel) {
    int32_t shunt_uV = 0;
    float current_A = 0;

    shunt_uV = get_shunt_voltage(channel);
    current_A = shunt_uV / (int32_t)_shuntRes[channel] / 1000.0;;
    return current_A;
}


float INA3221::get_voltage(ina3221_ch_t channel) {
    float voltage_V = 0.0;
```

```
    ina3221_reg_t reg;
    uint16_t val_raw = 0;

    switch(channel){
        case INA3221_CH1:
            reg = INA3221_REG_CH1_BUSV;
            break;
        case INA3221_CH2:
            reg = INA3221_REG_CH2_BUSV;
            break;
        case INA3221_CH3:
            reg = INA3221_REG_CH3_BUSV;
            break;
    }

    _read(reg, &val_raw);
    voltage_V = val_raw / 1000.0;
    return voltage_V;
}


// alerts
void INA3221::set_warn_alert_limit(ina3221_ch_t channel, float voltage_v) {
    ina3221_reg_t reg;
    uint16_t val = (uint16_t)(voltage_v * 1000); // Convert V to mV

    switch(channel) {
        case INA3221_CH1:
            reg = INA3221_REG_CH1_WARNING_ALERT_LIM;
            break;
        case INA3221_CH2:
            reg = INA3221_REG_CH2_WARNING_ALERT_LIM;
            break;
        case INA3221_CH3:
            reg = INA3221_REG_CH3_WARNING_ALERT_LIM;
            break;
    }
    _write(reg, &val);
}


void INA3221::set_crit_alert_limit(ina3221_ch_t channel, float voltage_v) {
    ina3221_reg_t reg;
    uint16_t val = (uint16_t)(voltage_v * 1000); // Convert V to mV

    switch(channel) {
        case INA3221_CH1:
            reg = INA3221_REG_CH1_CRIT_ALERT_LIM;
            break;
        case INA3221_CH2:
            reg = INA3221_REG_CH2_CRIT_ALERT_LIM;
            break;
        case INA3221_CH3:
            reg = INA3221_REG_CH3_CRIT_ALERT_LIM;
            break;
    }
    _write(reg, &val);
}


void INA3221::set_power_valid_limit(float voltage_upper_v, float voltage_lower_v) {
    uint16_t val;

    val = (uint16_t)(voltage_upper_v * 1000);
    _write(INA3221_REG_PWR_VALID_HI_LIM, &val);

    val = (uint16_t)(voltage_lower_v * 1000);
    _write(INA3221_REG_PWR_VALID_LO_LIM, &val);
}


void INA3221::enable_alerts() {
    masken_reg_t masken;
    _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);

    masken.warn_alert_ch1 = 1;
    masken.warn_alert_ch2 = 1;
    masken.warn_alert_ch3 = 1;
    masken.crit_alert_ch1 = 1;
    masken.crit_alert_ch2 = 1;
    masken.crit_alert_ch3 = 1;
    masken.pwr_valid_alert = 1;

    _write(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
}


```

```
bool INA3221::get_warn_alert(ina3221_ch_t channel) {
    masken_reg_t masken;
    _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);

    switch(channel) {
        case INA3221_CH1: return masken.warn_alert_ch1;
        case INA3221_CH2: return masken.warn_alert_ch2;
        case INA3221_CH3: return masken.warn_alert_ch3;
        default: return false;
    }
}


bool INA3221::get_crit_alert(ina3221_ch_t channel) {
    masken_reg_t masken;
    _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);

    switch(channel) {
        case INA3221_CH1: return masken.crit_alert_ch1;
        case INA3221_CH2: return masken.crit_alert_ch2;
        case INA3221_CH3: return masken.crit_alert_ch3;
        default: return false;
    }
}


bool INA3221::get_power_valid_alert() {
    masken_reg_t masken;
    _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
    return masken.pwr_valid_alert;
}


void INA3221::set_alert_latch(bool enable) {
    masken_reg_t masken;
    _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
    masken.warn_alert_latch_en = enable;
    masken.crit_alert_latch_en = enable;
    _write(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
}


// private
void INA3221::_read(ina3221_reg_t reg, uint16_t *val) {
    uint8_t reg_buf = reg;
    uint8_t data[2];

    int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, &reg_buf, 1, true);
    if (ret != 1) {
        std::cerr « "Failed to write register address to I2C device." « std::endl;
        return;
    }

    ret = i2c_read_blocking(MAIN_I2C_PORT, _i2c_addr, data, 2, false);
    if (ret != 2) {
        std::cerr « "Failed to read data from I2C device." « std::endl;
        return;
    }

    *val = (data[0] « 8) | data[1];
}


void INA3221::_write(ina3221_reg_t reg, uint16_t *val) {
    uint8_t buf[3];
    buf[0] = reg;
    buf[1] = (*val » 8) & 0xFF; // MSB
    buf[2] = (*val) & 0xFF;      // LSB

    int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, buf, 3, false);
    if (ret != 3) {
        std::cerr « "Failed to write data to I2C device." « std::endl;
    }
}
```

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_ALER , 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_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 = 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.

```
#ifndef BEASTDEVICES_INA3221_H
#define BEASTDEVICES_INA3221_H

#include <stdint.h>
#include <iostream>
#include <hardware/i2c.h>

typedef enum {
    INA3221_ADDR40_GND = 0b1000000, // A0 pin -> GND
    INA3221_ADDR41_VCC = 0b1000001, // A0 pin -> VCC
    INA3221_ADDR42_SDA = 0b1000010, // A0 pin -> SDA
    INA3221_ADDR43_SCL = 0b1000011  // A0 pin -> SCL
} ina3221_addr_t;

typedef enum {
    INA3221_CH1 = 0,
    INA3221_CH2,
    INA3221_CH3,
} ina3221_ch_t;

const int INA3221_CH_NUM = 3;
const int SHUNT_VOLTAGE_LSB_UV = 5;


typedef enum {
    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_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 = 0xFE,
    INA3221_REG_DIE_ID = 0xFF
} ina3221_reg_t;

typedef enum {
    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
} ina3221_conv_time_t;

typedef enum {
    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
} ina3221_avg_mode_t;

class INA3221 {

    typedef struct {
        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;
```

```
    } conf_reg_t;

    typedef struct {
        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;
    } masken_reg_t;

    i2c_inst_t* _i2c;
    // I2C address
    ina3221_addr_t _i2c_addr;

    // Shunt resistance in mOhm
    uint32_t _shuntRes[INA3221_CH_NUM];

    // Series filter resistance in Ohm
    uint32_t _filterRes[INA3221_CH_NUM];

    // Value of Mask/Enable register.
    masken_reg_t _masken_reg;

    // Reads 16 bytes from a register.
    void _read(ina3221_reg_t reg, uint16_t *val);

    // Writes 16 bytes to a register.
    void _write(ina3221_reg_t reg, uint16_t *val);

public:

    INA3221(ina3221_addr_t addr, i2c_inst_t* i2c);
    // Initializes INA3221
    bool begin();

    // Gets a register value.
    uint16_t read_register(ina3221_reg_t reg);

    // Resets INA3221
    void reset();

    // Sets operating mode to power-down
    void set_mode_power_down();

    // Sets operating mode to continious
    void set_mode_continuous();

    // Sets operating mode to triggered (single-shot)
    void set_mode_triggered();

    // Enables shunt-voltage measurement
    void set_shunt_measurement_enable();

    // Disables shunt-voltage mesurement
    void set_shunt_measurement_disable();

    // Enables bus-voltage measurement
    void set_bus_measurement_enable();

    // Disables bus-voltage measureement
    void set_bus_measurement_disable();

    // Sets averaging mode. Sets number of samples that are collected
    // and averaged togehter.
    void set_averaging_mode(ina3221_avg_mode_t mode);

    // Sets bus-voltage conversion time.
    void set_bus_conversion_time(ina3221_conv_time_t convTime);

    // Sets shunt-voltage conversion time.
    void set_shunt_conversion_time(ina3221_conv_time_t convTime);

    // Gets manufacturer ID.
    // Should read 0x5449.
    uint16_t get_manufacturer_id();

```

```
    // Gets die ID.
    // Should read 0x3220.
    uint16_t get_die_id();

    // Gets shunt voltage in uV.
    int32_t get_shunt_voltage(ina3221_ch_t channel);

    // Gets current in A.
    float get_current(ina3221_ch_t channel);

    float get_current_ma(ina3221_ch_t channel);

    // Gets bus voltage in V.
    float get_voltage(ina3221_ch_t channel);

    void set_warn_alert_limit(ina3221_ch_t channel, float voltage_v);
    void set_crit_alert_limit(ina3221_ch_t channel, float voltage_v);
    void set_power_valid_limit(float voltage_upper_v, float voltage_lower_v);
    void enable_alerts();
    bool get_warn_alert(ina3221_ch_t channel);
    bool get_crit_alert(ina3221_ch_t channel);
    bool get_power_valid_alert();
    void set_alert_latch(bool enable);
};

#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.

```
#include "PowerManager.h"
#include <iostream>
#include <sstream>
#include "event_manager.h"

PowerManager::PowerManager(i2c_inst_t* i2c)
    : ina3221_(INA3221_ADDR40_GND, i2c) {
        recursive_mutex_init(&powerman_mutex_);
    };

    bool PowerManager::initialize() {
        recursive_mutex_enter_blocking(&powerman_mutex_);
        initialized_ = ina3221_.begin();

        if (initialized_) {
            // Set up alerts
            ina3221_.set_warn_alert_limit(INA3221_CH2, VOLTAGE_LOW_THRESHOLD);
            ina3221_.set_crit_alert_limit(INA3221_CH2, VOLTAGE_OVERCHARGE_THRESHOLD);
            ina3221_.set_power_valid_limit(VOLTAGE_OVERCHARGE_THRESHOLD, VOLTAGE_LOW_THRESHOLD);
            ina3221_.enable_alerts();
            ina3221_.set_alert_latch(true);
        }

```

```
        recursive_mutex_exit(&powerman_mutex_);
        return initialized_;
    }

    std::string PowerManager::read_device_ids() {
        if (!initialized_) return "noinit";
        recursive_mutex_enter_blocking(&powerman_mutex_);
        std::stringstream man_ss;
        man_ss « std::hex « ina3221_.get_manufacturer_id();
        std::string MAN = "MAN 0x" + man_ss.str();

        std::stringstream die_ss;
        die_ss « std::hex « ina3221_.get_die_id();
        std::string DIE = "DIE 0x" + die_ss.str();
        recursive_mutex_exit(&powerman_mutex_);
        return MAN + " - " + DIE;
    }

float PowerManager::get_voltage_battery() {
    if (!initialized_) return 0.0f;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    float voltage = ina3221_.get_voltage(INA3221_CH1);
    recursive_mutex_exit(&powerman_mutex_);
    return voltage;
}

float PowerManager::get_voltage_5v() {
    if (!initialized_) return 0.0f;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    float voltage = ina3221_.get_voltage(INA3221_CH2);
    recursive_mutex_exit(&powerman_mutex_);
    return voltage;
}

float PowerManager::get_current_charge_usb() {
    if (!initialized_) return 0.0f;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    float current = ina3221_.get_current_ma(INA3221_CH1);
    recursive_mutex_exit(&powerman_mutex_);
    return current;
}

float PowerManager::get_current_draw() {
    if (!initialized_) return 0.0f;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    float current = ina3221_.get_current_ma(INA3221_CH2);
    recursive_mutex_exit(&powerman_mutex_);
    return current;
}

float PowerManager::get_current_charge_solar() {
    if (!initialized_) return 0.0f;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    float current = ina3221_.get_current_ma(INA3221_CH3);
    recursive_mutex_exit(&powerman_mutex_);
    return current;
}

float PowerManager::get_current_charge_total() {
    if (!initialized_) return 0.0f;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    float current = ina3221_.get_current_ma(INA3221_CH1) + ina3221_.get_current_ma(INA3221_CH3);
    recursive_mutex_exit(&powerman_mutex_);
    return current;
}

void PowerManager::configure(const std::map<std::string, std::string>& config) {
    if (!initialized_) return;
    recursive_mutex_enter_blocking(&powerman_mutex_);

    if (config.find("operating_mode") != config.end()) {
        if (config.at("operating_mode") == "continuous") {
            ina3221_.set_mode_continuous();
        }
    }

    if (config.find("averaging_mode") != config.end()) {
        int avg_mode = std::stoi(config.at("averaging_mode"));
        switch(avg_mode) {
            case 1:
                ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_1);
                break;
            case 4:
                ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_4);
                break;
            case 16:
                ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_16);
```

```
                break;
            default:
                ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_16);
        }
    }
    recursive_mutex_exit(&powerman_mutex_);
}

bool PowerManager::is_charging_solar() {
    if (!initialized_) return false;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    bool active = get_current_charge_solar() > SOLAR_CURRENT_THRESHOLD;
    recursive_mutex_exit(&powerman_mutex_);
    return active;
}

bool PowerManager::is_charging_usb() {
    if (!initialized_) return false;
    recursive_mutex_enter_blocking(&powerman_mutex_);
    bool connected = get_current_charge_usb() > USB_CURRENT_THRESHOLD;
    recursive_mutex_exit(&powerman_mutex_);
    return connected;
}

bool PowerManager::check_power_alerts() {
    if (!initialized_) return false;

    recursive_mutex_enter_blocking(&powerman_mutex_);

    bool status_changed = false;

    // Check warning alert (low battery)
    if (ina3221_.get_warn_alert(INA3221_CH2)) {
        EventEmitter::emit(EventGroup::POWER, PowerEvent::LOW_BATTERY);
        status_changed = true;
    }

    // Check critical alert (overcharge)
    if (ina3221_.get_crit_alert(INA3221_CH2)) {
        EventEmitter::emit(EventGroup::POWER, PowerEvent::OVERCHARGE);
        status_changed = true;
    }

    // Check power valid alert
    if (ina3221_.get_power_valid_alert()) {
        EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_NORMAL);
        status_changed = true;
    }

    recursive_mutex_exit(&powerman_mutex_);
    return status_changed;
}
```

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.

```
#ifndef POWER_MANAGER_H
#define POWER_MANAGER_H

#include "INA3221/INA3221.h"
#include <map>
#include <string>
#include <hardware/i2c.h>
#include "pico/stdlib.h"
#include "pico/mutex.h"

class PowerManager {
public:
    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 constexpr float SOLAR_CURRENT_THRESHOLD = 50.0f;  // mA
    static constexpr float USB_CURRENT_THRESHOLD = 50.0f;    // mA
    static constexpr float VOLTAGE_LOW_THRESHOLD = 4.6f;     // V
    static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f; // V
    static constexpr float FALL_RATE_THRESHOLD = -0.02f;     // V/sample
    static constexpr int FALLING_TREND_REQUIRED = 3;         // samples

```

```
private:
    INA3221 ina3221_;
    bool initialized_;
    recursive_mutex_t powerman_mutex_;
    bool charging_solar_active_ = false;
    bool charging_usb_active_ = false;
};

#endif // POWER_MANAGER_H
```

8.67 lib/sensors/BH1750/BH1750.cpp File Reference

```
#include "BH1750.h"
#include "pico/stdlib.h"
#include <stdio.h>
#include <iostream>
```

Include dependency graph for BH1750.cpp:

8.68 BH1750.cpp

Go to the documentation of this file.

```
#include "BH1750.h"
#include "pico/stdlib.h"
#include <stdio.h>
#include <iostream>

BH1750::BH1750(uint8_t addr) : _i2c_addr(addr) {}

bool BH1750::begin(Mode mode) {
    write8(static_cast<uint8_t>(Mode::POWER_ON));
    write8(static_cast<uint8_t>(Mode::RESET));
    configure(mode);
    configure(BH1750::Mode::POWER_ON);
    uint8_t check = 0;
    uint8_t cmd = 0x10; // Continuously H-Resolution Mode
    if (i2c_write_blocking(i2c1, _i2c_addr, &cmd, 1, false) == 1) {
        std::cout « "BH1750 sensor found at 0x" « std::hex « (int)_i2c_addr « std::endl;
        return true;
    }
    return false;
}

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

```
    uint8_t modeVal = static_cast<uint8_t>(mode);
    switch (mode) {
        case Mode::CONTINUOUS_HIGH_RES_MODE:
        case Mode::CONTINUOUS_HIGH_RES_MODE_2:
        case Mode::CONTINUOUS_LOW_RES_MODE:
        case Mode::ONE_TIME_HIGH_RES_MODE:
        case Mode::ONE_TIME_HIGH_RES_MODE_2:
        case Mode::ONE_TIME_LOW_RES_MODE:
            write8(modeVal);
            sleep_ms(10);
            break;
        default:
            printf("Invalid measurement mode\n");
            break;
    }
}

float BH1750::get_light_level() {
    uint8_t buffer[2];
    i2c_read_blocking(i2c_default, _i2c_addr, buffer, 2, false);
    uint16_t level = (buffer[0] << 8) | buffer[1];

    float lux = static_cast<float>(level) / 1.2f;
    return lux;
}

void BH1750::write8(uint8_t data) {
    uint8_t buf[1] = {data};
    i2c_write_blocking(i2c_default, _i2c_addr, buf, 1, false);
}
```

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             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") {
```

```
                sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE_2);
            }
            else if (value == "one_time_low_resolution") {
                sensor_.configure(BH1750::Mode::ONE_TIME_LOW_RES_MODE);
            }
            else {
                std::cerr « "[BH1750Wrapper] Unknown measurement_mode value: " « value « std::endl;
                return false;
            }
        }
        // Handle additional configuration keys here
        else {
            std::cerr « "[BH1750Wrapper] Unknown configuration key: " « key « std::endl;
            return false;
        }
    }
    return true;
}
```

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.

```
#ifndef BH1750_WRAPPER_H
#define BH1750_WRAPPER_H

#include "ISensor.h"
#include "BH1750.h"
#include <map>
#include <string>

class BH1750Wrapper : public ISensor {
private:
    BH1750 sensor_;
    bool initialized_ = false;

public:
    BH1750Wrapper();
    int get_i2c_addr();
    bool init() override;
    float read_data(SensorDataTypeIdentifier type) override;
    bool is_initialized() const override;
    SensorType get_type() const override;

    bool configure(const std::map<std::string, std::string>& config);

    uint8_t get_address() const override {
        return 0x23;
    }
};

#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     }
```

```

    initialized_ = true;
    std::cout « "BME280 sensor initialized_ successfully.\n";
    return true;
}

void BME280::reset() {
    uint8_t buf[2] = { REG_RESET, 0xB6 };
    int ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
    if (ret != 2) {
        std::cerr « "Failed to reset BME280 sensor.\n";
    }
    sleep_ms(10); // Wait for reset to complete
}

bool BME280::read_raw_all(int32_t* temperature, int32_t* pressure, int32_t* humidity) {
    if (!initialized_) {
        std::cerr « "BME280 not initialized_.\n";
        return false;
    }

    // Define the starting register address
    uint8_t start_reg = REG_PRESSURE_MSB;
    // Total bytes to read: 3 (pressure) + 3 (temperature) + 2 (humidity) = 8
    uint8_t buf[8] = {0};

    // Write the starting register address
    int ret = i2c_write_blocking(i2c_port, device_addr, &start_reg, 1, true);
    if (ret != 1) {
        std::cerr « "Failed to write starting register address to BME280.\n";
        return false;
    }

    // Read data
    ret = i2c_read_blocking(i2c_port, device_addr, buf, 8, false);
    if (ret != 8) {
        std::cerr « "Failed to read data from BME280.\n";
        return false;
    }

    // Combine bytes to form raw values
    *pressure = ((int32_t)buf[0] « 12) | ((int32_t)buf[1] « 4) | ((int32_t)(buf[2] » 4));
    *temperature = ((int32_t)buf[3] « 12) | ((int32_t)buf[4] « 4) | ((int32_t)(buf[5] » 4));
    *humidity = ((int32_t)buf[6] « 8) | (int32_t)buf[7];

    return true;
}

float BME280::convert_temperature(int32_t temp_raw) const {
    int32_t var1, var2;
    var1 = ((((temp_raw » 3) - ((int32_t)calib_params.dig_t1 « 1))) * ((int32_t)calib_params.dig_t2)) » 11;
    var2 = (((((temp_raw » 4) - ((int32_t)calib_params.dig_t1)) * ((temp_raw » 4) - ((int32_t)calib_params.dig_t1))) » 12) * ((int32_t)calib_params.dig_t3)) » 14;
    t_fine = var1 + var2;
    float T = (t_fine * 5 + 128) » 8;
    return T / 100.0f;
}

float BME280::convert_pressure(int32_t pressure_raw) const {
    int64_t var1, var2, p;
    var1 = ((int64_t)t_fine) - 128000;
    var2 = var1 * var1 * (int64_t)calib_params.dig_p6;
    var2 = var2 + ((var1 * (int64_t)calib_params.dig_p5) « 17);
    var2 = var2 + (((int64_t)calib_params.dig_p4) « 35);
    var1 = ((var1 * var1 * (int64_t)calib_params.dig_p3) » 8) + ((var1 * (int64_t)calib_params.dig_p2) « 12);
    var1 = ((((int64_t)1 « 47) + var1)) * ((int64_t)calib_params.dig_p1) » 33;

    if (var1 == 0) {
        return 0.0f; // avoid exception caused by division by zero
    }
    p = 1048576 - pressure_raw;
    p = (((p « 31) - var2) * 3125) / var1;
    var1 = (((int64_t)calib_params.dig_p9) * (p » 13) * (p » 13)) » 25;
    var2 = (((int64_t)calib_params.dig_p8) * p) » 19;

    p = ((p + var1 + var2) » 8) + (((int64_t)calib_params.dig_p7) « 4);
    return (float)p / 25600.0f; // in hPa
}

float BME280::convert_humidity(int32_t humidity_raw) const {
    int32_t v_x1_u32r;
    v_x1_u32r = t_fine - 76800;
    v_x1_u32r = (((((humidity_raw « 14) - ((int32_t)calib_params.dig_h4 « 20) - ((int32_t)calib_params.dig_h5 * v_x1_u32r)) + 16384) » 15) *
                 (((((((v_x1_u32r * (int32_t)calib_params.dig_h6) » 10) * (((v_x1_u32r *
```

```
    (int32_t)calib_params.dig_h3) » 11) + 32768)) » 10) + 2097152) *
                    (int32_t)calib_params.dig_h2 + 8192) » 14));
    v_x1_u32r = std::max(v_x1_u32r, (int32_t)0);
    v_x1_u32r = std::min(v_x1_u32r, (int32_t)419430400);
    float h = v_x1_u32r » 12;
    return h / 1024.0f;
}

bool BME280::get_calibration_parameters() {
    // Read temperature and pressure calibration data (0x88 to 0xA1)
    uint8_t calib_data[26];
    uint8_t reg = REG_DIG_T1_LSB;
    int ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
    if (ret != 1) {
        std::cerr « "Failed to write to BME280.\n";
        return false;
    }
    ret = i2c_read_blocking(i2c_port, device_addr, calib_data, 26, false);
    if (ret != 26) {
        std::cerr « "Failed to read calibration data from BME280.\n";
        return false;
    }

    // Parse temperature calibration data
    calib_params.dig_t1 = (uint16_t)(calib_data[1] « 8 | calib_data[0]);
    calib_params.dig_t2 = (int16_t)(calib_data[3] « 8 | calib_data[2]);
    calib_params.dig_t3 = (int16_t)(calib_data[5] « 8 | calib_data[4]);

    // Parse pressure calibration data
    calib_params.dig_p1 = (uint16_t)(calib_data[7] « 8 | calib_data[6]);
    calib_params.dig_p2 = (int16_t)(calib_data[9] « 8 | calib_data[8]);
    calib_params.dig_p3 = (int16_t)(calib_data[11] « 8 | calib_data[10]);
    calib_params.dig_p4 = (int16_t)(calib_data[13] « 8 | calib_data[12]);
    calib_params.dig_p5 = (int16_t)(calib_data[15] « 8 | calib_data[14]);
    calib_params.dig_p6 = (int16_t)(calib_data[17] « 8 | calib_data[16]);
    calib_params.dig_p7 = (int16_t)(calib_data[19] « 8 | calib_data[18]);
    calib_params.dig_p8 = (int16_t)(calib_data[21] « 8 | calib_data[20]);
    calib_params.dig_p9 = (int16_t)(calib_data[23] « 8 | calib_data[22]);

    calib_params.dig_h1 = calib_data[25];

    // Read humidity calibration data (0xE1 to 0xE7)
    reg = 0xE1;
    ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
    if (ret != 1) {
        std::cerr « "Failed to write to BME280 for humidity calibration.\n";
        return false;
    }

    uint8_t hum_calib_data[7];
    ret = i2c_read_blocking(i2c_port, device_addr, hum_calib_data, 7, false);
    if (ret != 7) {
        std::cerr « "Failed to read humidity calibration data from BME280.\n";
        return false;
    }

    // Parse humidity calibration data
    calib_params.dig_h2 = (int16_t)(hum_calib_data[1] « 8 | hum_calib_data[0]);
    calib_params.dig_h3 = hum_calib_data[2];
    calib_params.dig_h4 = (int16_t)((hum_calib_data[3] « 4) | (hum_calib_data[4] & 0x0F));
    calib_params.dig_h5 = (int16_t)((hum_calib_data[5] « 4) | (hum_calib_data[4] » 4));
    calib_params.dig_h6 = (int8_t)hum_calib_data[6];

    return true;
}

bool BME280::configure_sensor() {
    uint8_t buf[2];

    // Set humidity oversampling (must be set before ctrl_meas)
    buf[0] = REG_CTRL_HUM;
    buf[1] = 0x05; // Humidity oversampling x16
    int ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
    if (ret != 2) {
        std::cerr « "Failed to write CTRL_HUM to BME280.\n";
        return false;
    }

    // Write config register
    buf[0] = REG_CONFIG;
    buf[1] = 0x00; // Default settings
    ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
    if (ret != 2) {
        std::cerr « "Failed to write CONFIG to BME280.\n";
        return false;
    }

```

```
    // Write ctrl_meas register
    buf[0] = REG_CTRL_MEAS;
    buf[1] = 0xB7; // Temp and pressure oversampling x16, normal mode
    ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
    if (ret !=2) {
        std::cerr « "Failed to write CTRL_MEAS to BME280.\n";
        return false;
    }

    return true;
}
```

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.

```
// BME280.h

#ifndef BME280_H
#define BME280_H

#include <cstdint>
#include <iostream>
#include "hardware/i2c.h"

// Calibration parameters structure
struct BME280CalibParam {
    // Temperature parameters
    uint16_t dig_t1;
    int16_t  dig_t2;
    int16_t  dig_t3;

    // Pressure parameters
    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;

    // Humidity parameters
    uint8_t  dig_h1;
    int16_t  dig_h2;
    uint8_t  dig_h3;
    int16_t  dig_h4;
    int16_t  dig_h5;
    int8_t   dig_h6;
};

// BME280 Class Definition
class BME280 {
public:
    // I2C Address Options
    static constexpr uint8_t ADDR_SDO_LOW = 0x76;
    static constexpr uint8_t ADDR_SDO_HIGH = 0x77;

    // Constructor
    BME280(i2c_inst_t* i2cPort, uint8_t address = ADDR_SDO_LOW);

    // Initialize the sensor
    bool init();

    // Reset the sensor
    void reset();

    // Read all raw data: temperature, pressure, and humidity
    bool read_raw_all(int32_t* temperature, int32_t* pressure, int32_t* humidity);

    // Convert raw data to actual values
    float convert_temperature(int32_t temp_raw) const;
    float convert_pressure(int32_t pressure_raw) const;
    float convert_humidity(int32_t humidity_raw) const;

private:
    // Configure the sensor
    bool configure_sensor();

    // Retrieve calibration parameters from the sensor
    bool get_calibration_parameters();

    // I2C port and device address
    i2c_inst_t* i2c_port;
    uint8_t device_addr;

```

```
    // Calibration parameters
    BME280CalibParam calib_params;

    // Initialization status
    bool initialized_;

    // Fine temperature parameter needed for compensation
    mutable int32_t t_fine;

    // Register Definitions
    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;

    // Calibration Registers
    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;

    // Humidity Calibration Registers
    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;

    // Number of calibration parameters to read
    static constexpr size_t NUM_CALIB_PARAMS = 24;
};

#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.

```
#include "BME280_WRAPPER.h"

BME280Wrapper::BME280Wrapper(i2c_inst_t* i2c) : sensor_(i2c) {}

bool BME280Wrapper::init() {
    initialized_ = sensor_.init();
    return initialized_;
}

float BME280Wrapper::read_data(SensorDataTypeIdentifier type) {
    int32_t temp_raw, pressure_raw, humidity_raw;
    sensor_.read_raw_all(&temp_raw, &pressure_raw, &humidity_raw);

    switch(type) {
        case SensorDataTypeIdentifier::TEMPERATURE:
            return sensor_.convert_temperature(temp_raw);
        case SensorDataTypeIdentifier::PRESSURE:
            return sensor_.convert_pressure(pressure_raw);
        case SensorDataTypeIdentifier::HUMIDITY:
            return sensor_.convert_humidity(humidity_raw);
        default:
            return 0.0f;
    }
}

bool BME280Wrapper::is_initialized() const {
    return initialized_;
}

SensorType BME280Wrapper::get_type() const {
    return SensorType::ENVIRONMENT;
}

bool BME280Wrapper::configure(const std::map<std::string, std::string>& config) {
    return true;
}
```

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
```

```
class BME280Wrapper : public ISensor {
private:
    BME280 sensor_;
    bool initialized_ = false;

public:
    BME280Wrapper(i2c_inst_t* i2c);

    bool init() override;
    float read_data(SensorDataTypeIdentifier type) override;
    bool is_initialized() const override;
    SensorType get_type() const override;
    bool configure(const std::map<std::string, std::string>& config) override;

    uint8_t get_address() const override {
        return 0x76;
    }

};

#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.

```
#include "HMC5883L.h"

HMC5883L::HMC5883L(i2c_inst_t* i2c, uint8_t address) : i2c(i2c), address(address) {}

bool HMC5883L::init() {
    // Continuous measurement mode, 15Hz data output rate
    if (!write_register(0x00, 0x70)) return false;
    if (!write_register(0x01, 0x20)) return false;
    if (!write_register(0x02, 0x00)) return false;
    return true;
}

bool HMC5883L::read(int16_t& x, int16_t& y, int16_t& z) {
```

```
    uint8_t buffer[6];
    if (!read_register(0x03, buffer, 6)) return false;

    x = (buffer[0] « 8) | buffer[1];
    z = (buffer[2] « 8) | buffer[3];
    y = (buffer[4] « 8) | buffer[5];

    if (x > 32767) x -= 65536;
    if (y > 32767) y -= 65536;
    if (z > 32767) z -= 65536;

    return true;
}

bool HMC5883L::write_register(uint8_t reg, uint8_t value) {
    uint8_t buffer[2] = {reg, value};
    return i2c_write_blocking(i2c, address, buffer, 2, false) == 2;
}

bool HMC5883L::read_register(uint8_t reg, uint8_t* buffer, size_t length) {
    if (i2c_write_blocking(i2c, address, &reg, 1, true) != 1) return false;
    return i2c_read_blocking(i2c, address, buffer, length, false) == length;
}
```

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.

```
#ifndef HMC5883L_H
#define HMC5883L_H

#include "hardware/i2c.h"

class HMC5883L {
public:
    HMC5883L(i2c_inst_t* i2c, uint8_t address = 0x0D);
    bool init();
    bool read(int16_t& x, int16_t& y, int16_t& z);

private:
    i2c_inst_t* i2c;
    uint8_t address;

    bool write_register(uint8_t reg, uint8_t value);
    bool read_register(uint8_t reg, uint8_t* buffer, size_t length);
};

#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.

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

HMC5883LWrapper::HMC5883LWrapper(i2c_inst_t* i2c) : sensor_(i2c), initialized_(false) {}

bool HMC5883LWrapper::init() {
    initialized_ = sensor_.init();
    return initialized_;
}

float HMC5883LWrapper::read_data(SensorDataTypeIdentifier type) {
    if (!initialized_) return 0.0f;

    int16_t x, y, z;
    if (!sensor_.read(x, y, z)) return 0.0f;

    const float LSB_TO_UT = 100.0 / 1090.0;
    float x_uT = x * LSB_TO_UT;
    float y_uT = y * LSB_TO_UT;
    float z_uT = z * LSB_TO_UT;

    switch (type) {
        case SensorDataTypeIdentifier::MAG_FIELD_X:
            return x_uT;
        case SensorDataTypeIdentifier::MAG_FIELD_Y:
            return y_uT;
        case SensorDataTypeIdentifier::MAG_FIELD_Z:
            return z_uT;
        default:
            return 0.0f;
    }
}

bool HMC5883LWrapper::is_initialized() const {
    return initialized_;
}

SensorType HMC5883LWrapper::get_type() const {
    return SensorType::MAGNETOMETER;
```

```
}

bool HMC5883LWrapper::configure(const std::map<std::string, std::string>& config) {
    // Configuration logic if needed
    return true;
}
```

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.

```
#ifndef HMC5883L_WRAPPER_H
#define HMC5883L_WRAPPER_H

#include "ISensor.h"
#include "HMC5883L.h"

class HMC5883LWrapper : public ISensor {
public:
    HMC5883LWrapper(i2c_inst_t* i2c);
    bool init() override;
    float read_data(SensorDataTypeIdentifier type) override;
    bool is_initialized() const override;
    SensorType get_type() const override;
    bool configure(const std::map<std::string, std::string>& config) override;

    uint8_t get_address() const override {
        return 0x0D;
    }

private:
    HMC5883L sensor_;
    bool initialized_;
};

#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 "lib/utils.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.

```
// ISensor.cpp
#include "ISensor.h"
#include "lib/sensors/BH1750/BH1750_WRAPPER.h"
#include "lib/sensors/BME280/BME280_WRAPPER.h"
#include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
#include "lib/utils.h"


SensorWrapper& SensorWrapper::get_instance() {
    static SensorWrapper instance;
    return instance;
}


SensorWrapper::SensorWrapper() = default;


bool SensorWrapper::sensor_init(SensorType type, i2c_inst_t* i2c) {
    switch(type) {
        case SensorType::LIGHT:
            sensors[type] = new BH1750Wrapper();
            break;
        case SensorType::ENVIRONMENT:
            sensors[type] = new BME280Wrapper(i2c);
            break;
        case SensorType::IMU:
            //sensors[type] = new MPU6050Wrapper(i2c);
            break;
        case SensorType::MAGNETOMETER:
            sensors[type] = new HMC5883LWrapper(i2c);
            break;
    }
    return sensors[type]->init();
}


bool SensorWrapper::sensor_configure(SensorType type, const std::map<std::string, std::string>&
config) {
    auto it = sensors.find(type);
    if (it != sensors.end() && it->second->is_initialized()) {
        return it->second->configure(config);
    }
    std::cerr « "Sensor not initialized or not found: " « static_cast<int>(type) « std::endl;
    return false;
}


float SensorWrapper::sensor_read_data(SensorType sensorType, SensorDataTypeIdentifier dataType) {
    auto it = sensors.find(sensorType);
    if (it != sensors.end() && it->second->is_initialized()) {
        return it->second->read_data(dataType);
    }
    return 0.0f;
}


std::vector<std::pair<SensorType, uint8_t» SensorWrapper::get_available_sensors() {
    std::vector<std::pair<SensorType, uint8_t» available_sensors;

    for (const auto& sensor_pair : sensors) {
        if (sensor_pair.second->is_initialized()) {
            available_sensors.push_back({sensor_pair.first, sensor_pair.second->get_address()});
        }
    }

    return available_sensors;
}


std::vector<std::pair<SensorType, uint8_t» SensorWrapper::scan_connected_sensors(i2c_inst_t* i2c) {
    std::vector<std::pair<SensorType, uint8_t» connected_sensors;

    // Define the address ranges to check for each sensor type
    struct SensorAddressInfo {
        SensorType type;
        std::vector<uint8_t> addresses;
    };

    std::vector<SensorAddressInfo> sensor_addresses = {
        {SensorType::LIGHT, {0x23, 0x5C}},              // BH1750 addresses
        {SensorType::ENVIRONMENT, {0x76, 0x77}},        // BME280 addresses
```

```
        {SensorType::MAGNETOMETER, {0x0D, 0x1E}},       // HMC5883L addresses
        {SensorType::IMU, {0x68, 0x69}}                 // MPU6050 addresses
    };

    // Buffer for receiving ACK/NACK
    uint8_t rxdata;

    for (const auto& sensor_info : sensor_addresses) {
        for (uint8_t addr : sensor_info.addresses) {
            // Try to read a byte from the device to see if it ACKs
            int result = i2c_read_blocking(i2c, addr, &rxdata, 1, false);
            if (result >= 0) {
                // We received an ACK, so the device exists
                connected_sensors.push_back({sensor_info.type, addr});
                uart_print("Found sensor at address 0x" + std::to_string(addr),
VerbosityLevel::DEBUG);
            }
        }
    }

    return connected_sensors;
}

```

8.93 lib/sensors/ISensor.h File Reference

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

Include dependency graph for ISensor.h:

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

Classes

- class ISensor
- class SensorWrapper

  Manages different sensor types and provides a unified interface for accessing sensor data.

Enumerations

- enum class SensorType { LIGHT , ENVIRONMENT , MAGNETOMETER , IMU }
- enum class SensorDataTypeIdentifier { 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 11 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 18 of file ISensor.h.

8.94 ISensor.h

Go to the documentation of this file.

```
// ISensor.h
#ifndef ISENSOR_H
#define ISENSOR_H

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

enum class SensorType {
    LIGHT,          // BH1750
    ENVIRONMENT,    // BME280
    MAGNETOMETER,   // HMC5883L
    IMU             // MPU6050
};

enum class SensorDataTypeIdentifier {
    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
};

class ISensor {
public:
    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;
    virtual uint8_t get_address() const = 0;
};

class SensorWrapper {
public:
    static SensorWrapper& get_instance();
    bool sensor_init(SensorType type, i2c_inst_t* i2c = nullptr);
    bool sensor_configure(SensorType type, const std::map<std::string, std::string>& config);
    float sensor_read_data(SensorType sensorType, SensorDataTypeIdentifier dataType);
    ISensor* get_sensor(SensorType type);

    std::vector<std::pair<SensorType, uint8_t»> scan_connected_sensors(i2c_inst_t* i2c);
    std::vector<std::pair<SensorType, uint8_t»> get_available_sensors();

private:
    std::map<SensorType, ISensor*> sensors;
    SensorWrapper();
};

#endif // ISENSOR_H
```

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

8.96 MPU6050.cpp

Go to the documentation of this file.

```
```

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>
```

```
#include <string>

class MPU6050Wrapper : public ISensor {
private:
    MPU6050 sensor_;
    bool initialized_ = false;

public:
    MPU6050Wrapper();

    bool init() override;
    float read_data(SensorDataTypeIdentifier type) override;
    bool is_initialized() const override;
    SensorType get_type() const override;

    bool configure(const std::map<std::string, std::string>& config);
};

#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 status_string;
00038     int err = fs_mount("/", fat, sd);
00039     if (err == -1) {
00040         status_string = "Formatting / with FAT";
00041         uart_print(status_string, VerbosityLevel::WARNING);
00042         err = fs_format(fat, sd);
00043         if (err == -1) {
00044             status_string = "fs_format error: " + std::string(strerror(errno));
00045             uart_print(status_string, VerbosityLevel::ERROR);
00046             return false;
00047         }
00048         err = fs_mount("/", fat, sd);
00049         if (err == -1) {
00050             status_string = "fs_mount error: " + std::string(strerror(errno));
00051             uart_print(status_string, 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.

```
#ifndef STORAGE_H
#define STORAGE_H

#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"


extern bool sd_card_mounted;

struct FileHandle {
    int fd;
    bool is_open;
};

bool fs_init(void);
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);

#endif
```

```

// void example_file_operations() {
//     // Open a file for writing
//     FileHandle log_file = fs_open_file("/log.txt", "w");
//     if (!log_file.is_open) {
//         uartPrint("Failed to open log file");
//         return;
//     }

//     // Write some data
//     const char* message = "Hello, World!\n";
//     ssize_t written = fs_write_file(log_file, message, strlen(message));
//     if (written < 0) {
//         uartPrint("Failed to write to log file");
//     }

//     // Close the file
//     fs_close_file(log_file);

//     // Open file for reading
//     log_file = fs_open_file("/log.txt", "r");
//     if (!log_file.is_open) {
//         uartPrint("Failed to open log file for reading");
//         return;
//     }

//     // Read the data
//     char buffer[128];
//     ssize_t bytes_read = fs_read_file(log_file, buffer, sizeof(buffer) - 1);
//     if (bytes_read > 0) {
//         buffer[bytes_read] = '\0';  // Null terminate the string
//         uartPrint(buffer);
//     }

//     // Close the file
//     fs_close_file(log_file);
// }
```

8.107 lib/telemetry/telemetry_manager.cpp File Reference

Implementation of telemetry collection and storage functionality.

```
#include "telemetry_manager.h"
#include "utils.h"
#include "storage.h"
#include "PowerManager.h"
#include "ISensor.h"
#include "DS3231.h"
#include <deque>
#include <mutex>
#include <iomanip>
#include <sstream>
#include <cstdio>
#include "communication.h"
```

Include dependency graph for telemetry_manager.cpp:

Macros

- #define TELEMETRY_CSV_PATH "/telemetry.csv"

Path to the telemetry CSV file on storage media.

- #define SENSOR_DATA_CSV_PATH "/sensors.csv"

  Path to the sensor data CSV file on storage media.
- #define DEFAULT_SAMPLE_INTERVAL_MS 2000

  Default interval between telemetry samples in milliseconds (2 seconds)
- #define DEFAULT_FLUSH_THRESHOLD 10

  Default number of records to collect before flushing to storage.

Functions

- bool telemetry_init ()

  Initialize the telemetry system.
- bool collect_telemetry ()

  Collect telemetry data from sensors and power subsystems.
- bool flush_telemetry ()

  Save buffered telemetry data to storage.
- bool flush_sensor_data ()

  Save buffered sensor data to storage.
- bool is_telemetry_collection_time (uint32_t current_time, uint32_t &last_collection_time)

  Check if it's time to collect telemetry based on interval.
- bool is_telemetry_flush_time (uint32_t &collection_counter)

  Check if it's time to flush telemetry buffer based on count.
- uint32_t get_telemetry_sample_interval ()

  Get the current sample interval in milliseconds.
- void set_telemetry_sample_interval (uint32_t interval_ms)

  Set the telemetry sample interval.
- uint32_t get_telemetry_flush_threshold ()

  Get the number of records before flushing to storage.
- void set_telemetry_flush_threshold (uint32_t records)

  Set the number of records before flushing to storage.
- std::string get_last_telemetry_record_csv ()

  Gets the last telemetry record as a CSV string.
- std::string get_last_sensor_record_csv ()

  Gets the last sensor data record as a CSV string.

Variables

- PowerManager powerManager
- DS3231 systemClock
- NMEAData nmea_data
- static uint32_t sample_interval_ms = DEFAULT_SAMPLE_INTERVAL_MS

  Current sampling interval in milliseconds.
- static uint32_t flush_threshold = DEFAULT_FLUSH_THRESHOLD

  Current flush threshold (number of records that triggers a flush)
- TelemetryRecord telemetry_buffer [TELEMETRY_BUFFER_SIZE]

  Circular buffer for telemetry records.
- size_t telemetry_buffer_count = 0
- size_t telemetry_buffer_write_index = 0
- SensorDataRecord sensor_data_buffer [TELEMETRY_BUFFER_SIZE]

  Circular buffer for sensor data records.
- size_t sensor_data_buffer_count = 0
- size_t sensor_data_buffer_write_index = 0
- mutex_t telemetry_mutex

  Mutex for thread-safe access to the telemetry buffer.

8.107.1 Detailed Description

Implementation of telemetry collection and storage functionality.

Handles collecting, buffering, and persisting telemetry data from various satellite subsystems including power, sensors, and GPS

Definition in file telemetry_manager.cpp.

8.107.2 Macro Definition Documentation

8.107.2.1 TELEMETRY_CSV_PATH

```
#define TELEMETRY_CSV_PATH "/telemetry.csv"
```

Path to the telemetry CSV file on storage media.

Definition at line 30 of file telemetry_manager.cpp.

8.107.2.2 SENSOR_DATA_CSV_PATH

```
#define SENSOR_DATA_CSV_PATH "/sensors.csv"
```

Path to the sensor data CSV file on storage media.

Definition at line 35 of file telemetry_manager.cpp.

8.107.2.3 DEFAULT_SAMPLE_INTERVAL_MS

```
#define DEFAULT_SAMPLE_INTERVAL_MS 2000
```

Default interval between telemetry samples in milliseconds (2 seconds)

Definition at line 40 of file telemetry_manager.cpp.

8.107.2.4 DEFAULT_FLUSH_THRESHOLD

```
#define DEFAULT_FLUSH_THRESHOLD 10
```

Default number of records to collect before flushing to storage.

Definition at line 45 of file telemetry_manager.cpp.

8.107.3 Variable Documentation

8.107.3.1 powerManager

```
PowerManager powerManager  [extern]
```

8.107.3.2 systemClock

```
DS3231 systemClock [extern]
```

8.107.3.3 nmea_data

```
NMEAData nmea_data [extern]
```

Definition at line 3 of file NMEA_data.cpp.

8.107.3.4 sample_interval_ms

```
uint32_t sample_interval_ms = DEFAULT_SAMPLE_INTERVAL_MS [static]
```

Current sampling interval in milliseconds.

Definition at line 50 of file telemetry_manager.cpp.

8.107.3.5 flush_threshold

```
uint32_t flush_threshold = DEFAULT_FLUSH_THRESHOLD [static]
```

Current flush threshold (number of records that triggers a flush)

Definition at line 55 of file telemetry_manager.cpp.

8.108 telemetry_manager.cpp

Go to the documentation of this file.

```
00001 #include "telemetry_manager.h"
00002 #include "utils.h"
00003 #include "storage.h"
00004 #include "PowerManager.h"
00005 #include "ISensor.h"
00006 #include "DS3231.h"
00007 #include <deque>
00008 #include <mutex>
00009 #include <iomanip>
00010 #include <sstream>
00011 #include <cstdio>
00012 #include "communication.h"
00013
00020
00021
00022 extern PowerManager powerManager;
00023 extern DS3231 systemClock;
00024 extern NMEAData nmea_data;
00025
00026
00030 #define TELEMETRY_CSV_PATH "/telemetry.csv"
00031
00035 #define SENSOR_DATA_CSV_PATH "/sensors.csv"
00036
00040 #define DEFAULT_SAMPLE_INTERVAL_MS 2000
00041
00045 #define DEFAULT_FLUSH_THRESHOLD 10
00046
00050 static uint32_t sample_interval_ms = DEFAULT_SAMPLE_INTERVAL_MS;
00051
00055 static uint32_t flush_threshold = DEFAULT_FLUSH_THRESHOLD;
```

```

TelemetryRecord telemetry_buffer[TELEMETRY_BUFFER_SIZE];
size_t telemetry_buffer_count = 0;
size_t telemetry_buffer_write_index = 0;

SensorDataRecord sensor_data_buffer[TELEMETRY_BUFFER_SIZE];
size_t sensor_data_buffer_count = 0;
size_t sensor_data_buffer_write_index = 0;

mutex_t telemetry_mutex;

bool telemetry_init() {
    mutex_init(&telemetry_mutex);

    // Create CSV files with headers if they don't exist
    if (sd_card_mounted) {
        bool success = true;

        // Create telemetry CSV file
        FILE* file = fopen(TELEMETRY_CSV_PATH, "w");
        if (!file) {
            file = fopen(TELEMETRY_CSV_PATH, "w");
            if (file) {
                fprintf(file, "timestamp,build,battery_v,system_v,usb_ma,solar_ma,discharge_ma,"

"gps_time,latitude,lat_dir,longitude,lon_dir,speed_knots,course_deg,date,"
                                "fix_quality,satellites,altitude_m\n");
                fclose(file);
                uart_print("Created new telemetry log", VerbosityLevel::INFO);
            } else {
                uart_print("Failed to create telemetry log", VerbosityLevel::ERROR);
                success = false;
            }
        } else {
            fclose(file);
        }

        // Create sensor data CSV file
        file = fopen(SENSOR_DATA_CSV_PATH, "w");
        if (!file) {
            file = fopen(SENSOR_DATA_CSV_PATH, "w");
            if (file) {
                fprintf(file, "timestamp,temperature,pressure,humidity,light\n");
                fclose(file);
                uart_print("Created new sensor data log", VerbosityLevel::INFO);
            } else {
                uart_print("Failed to create sensor data log", VerbosityLevel::ERROR);
                success = false;
            }
        } else {
            fclose(file);
        }

        return success;
    }

    uart_print("Telemetry system initialized (storage not available)", VerbosityLevel::WARNING);
    return false;
}

bool collect_telemetry() {
    SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();

    TelemetryRecord record;
    record.timestamp = systemClock.get_unix_time();
    record.build_version = std::to_string(BUILD_NUMBER);

    // Power data
    record.battery_voltage = powerManager.get_voltage_battery();
    record.system_voltage = powerManager.get_voltage_5v();
    record.charge_current_usb = powerManager.get_current_charge_usb();
    record.charge_current_solar = powerManager.get_current_charge_solar();
    record.discharge_current = powerManager.get_current_draw();

    // Get GPS RMC data
    std::vector<std::string> rmc_tokens = nmea_data.get_rmc_tokens();
    if (rmc_tokens.size() >= 12) {  // RMC has at least 12 fields when complete
        record.time = rmc_tokens[1];
        record.latitude = rmc_tokens[3];
        record.lat_dir = rmc_tokens[4];
        record.longitude = rmc_tokens[5];
        record.lon_dir = rmc_tokens[6];
        record.speed = rmc_tokens[7];
        record.course = rmc_tokens[8];
        record.date = rmc_tokens[9];
    } else {
        // Fill with defaults if no GPS data
```

```
        record.time = "";
        record.latitude = "";
        record.lat_dir = "";
        record.longitude = "";
        record.lon_dir = "";
        record.speed = "";
        record.course = "";
        record.date = "";
    }

    // Get GPS GGA data
    std::vector<std::string> gga_tokens = nmea_data.get_gga_tokens();
    if (gga_tokens.size() >= 15) {  // GGA has 15 fields when complete
        record.fix_quality = gga_tokens[6];
        record.satellites = gga_tokens[7];
        record.altitude = gga_tokens[9];
    } else {
        // Fill with defaults if no GPS data
        record.fix_quality = "";
        record.satellites = "";
        record.altitude = "";
    }

    SensorDataRecord sensor_record;
    sensor_record.timestamp = systemClock.get_unix_time();
    sensor_record.temperature = sensor_wrapper.sensor_read_data(SensorType::ENVIRONMENT,
SensorDataTypeIdentifier::TEMPERATURE);
    sensor_record.pressure = sensor_wrapper.sensor_read_data(SensorType::ENVIRONMENT,
SensorDataTypeIdentifier::PRESSURE);
    sensor_record.humidity = sensor_wrapper.sensor_read_data(SensorType::ENVIRONMENT,
SensorDataTypeIdentifier::HUMIDITY);
    sensor_record.light = sensor_wrapper.sensor_read_data(SensorType::LIGHT,
SensorDataTypeIdentifier::LIGHT_LEVEL);

    mutex_enter_blocking(&telemetry_mutex);

    telemetry_buffer[telemetry_buffer_write_index] = record;
    sensor_data_buffer[telemetry_buffer_write_index] = sensor_record;
    telemetry_buffer_write_index = (telemetry_buffer_write_index + 1) % TELEMETRY_BUFFER_SIZE;
    if (telemetry_buffer_count < TELEMETRY_BUFFER_SIZE) {
        telemetry_buffer_count++;
    }

    mutex_exit(&telemetry_mutex);

    uart_print("Telemetry collected", VerbosityLevel::DEBUG);

    return true;
}

bool flush_telemetry() {
    if (!sd_card_mounted) {
        return false;
    }


    mutex_enter_blocking(&telemetry_mutex);

    if (telemetry_buffer_count == 0) {
        mutex_exit(&telemetry_mutex);
        return true; // Nothing to save
    }

    FILE* file = fopen(TELEMETRY_CSV_PATH, "a");
    if (!file) {
        uart_print("Failed to open telemetry log for writing", VerbosityLevel::ERROR);
        mutex_exit(&telemetry_mutex);
        return false;
    }

    // Calculate start index (for circular buffer)
    size_t read_index = 0;
    if (telemetry_buffer_count == TELEMETRY_BUFFER_SIZE) {
        // Buffer is full, start from oldest entry
        read_index = telemetry_buffer_write_index;
    }

    // Write all records to CSV
    for (size_t i = 0; i < telemetry_buffer_count; i++) {
        fprintf(file, "%s\n", telemetry_buffer[read_index].to_csv().c_str());
        read_index = (read_index + 1) % TELEMETRY_BUFFER_SIZE;
    }

    // Clear buffer after successful write
    telemetry_buffer_count = 0;
    telemetry_buffer_write_index = 0;

```

```
    fclose(file);

    mutex_exit(&telemetry_mutex);
    return true;
}

bool flush_sensor_data() {
    if (!sd_card_mounted) {
        return false;
    }

    mutex_enter_blocking(&telemetry_mutex);

    if (telemetry_buffer_count == 0) {
        mutex_exit(&telemetry_mutex);
        return true; // Nothing to save
    }

    FILE* file = fopen(SENSOR_DATA_CSV_PATH, "a");
    if (!file) {
        uart_print("Failed to open sensor data log for writing", VerbosityLevel::ERROR);
        mutex_exit(&telemetry_mutex);
        return false;
    }

    // Calculate start index (for circular buffer)
    size_t read_index = 0;
    if (telemetry_buffer_count == TELEMETRY_BUFFER_SIZE) {
        // Buffer is full, start from oldest entry
        read_index = telemetry_buffer_write_index;
    }

    // Write all records to CSV
    for (size_t i = 0; i < telemetry_buffer_count; i++) {
        fprintf(file, "%s\n", sensor_data_buffer[read_index].to_csv().c_str());
        read_index = (read_index + 1) % TELEMETRY_BUFFER_SIZE;
    }

    // We don't need to clear the buffer here since it's cleared in flush_telemetry()
    // and both buffers use the same indices

    fclose(file);

    mutex_exit(&telemetry_mutex);
    return true;
}

bool is_telemetry_collection_time(uint32_t current_time, uint32_t& last_collection_time) {
    if (current_time - last_collection_time >= sample_interval_ms) {
        last_collection_time = current_time;
        return true;
    }
    return false;
}

bool is_telemetry_flush_time(uint32_t& collection_counter) {
    if (collection_counter >= flush_threshold) {
        collection_counter = 0;
        return true;
    }
    return false;
}

uint32_t get_telemetry_sample_interval() {
    return sample_interval_ms;
}

void set_telemetry_sample_interval(uint32_t interval_ms) {
    if (interval_ms >= 100) { // Minimum 100ms
        sample_interval_ms = interval_ms;
    }
}

uint32_t get_telemetry_flush_threshold() {
    return flush_threshold;
}

void set_telemetry_flush_threshold(uint32_t records) {
    if (records >= 1 && records <= 100) {
        flush_threshold = records;
    }
}

// Remove the template function and implement the two specific functions directly

std::string get_last_telemetry_record_csv() {
    mutex_enter_blocking(&telemetry_mutex);
```

```
    if (telemetry_buffer_count == 0) {
        mutex_exit(&telemetry_mutex);
        return "";
    }

    size_t last_record_index = (telemetry_buffer_write_index + TELEMETRY_BUFFER_SIZE - 1) %
TELEMETRY_BUFFER_SIZE;

    TelemetryRecord last_record = telemetry_buffer[last_record_index];

    mutex_exit(&telemetry_mutex);

    return last_record.to_csv();
}

std::string get_last_sensor_record_csv() {
    mutex_enter_blocking(&telemetry_mutex);

    if (telemetry_buffer_count == 0) {
        mutex_exit(&telemetry_mutex);
        return "";
    }

    size_t last_record_index = (telemetry_buffer_write_index + TELEMETRY_BUFFER_SIZE - 1) %
TELEMETRY_BUFFER_SIZE;

    SensorDataRecord last_record = sensor_data_buffer[last_record_index];

    mutex_exit(&telemetry_mutex);

    return last_record.to_csv();
}
```

8.109 lib/telemetry/telemetry_manager.h File Reference

System telemetry collection and logging.

```
#include <cstdint>
#include <string>
#include "pico/stdlib.h"
#include "lib/location/NMEA/nmea_data.h"
#include "utils.h"
#include "storage.h"
#include "PowerManager.h"
#include "ISensor.h"
#include "DS3231.h"
#include <deque>
#include <mutex>
#include <iomanip>
#include <sstream>
#include <cstdio>
#include "communication.h"
#include <functional>
```

Include dependency graph for telemetry_manager.h:

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

Classes

- struct TelemetryRecord

  Structure representing a single telemetry data point.
- struct SensorDataRecord

Functions

- bool telemetry_init ()

  Initialize the telemetry system.
- bool collect_telemetry ()

  Collect telemetry data from sensors and power subsystems.
- bool flush_telemetry ()

  Save buffered telemetry data to storage.
- bool flush_sensor_data ()

  Save buffered sensor data to storage.
- bool is_telemetry_collection_time (uint32_t current_time, uint32_t &last_collection_time)

  Check if it's time to collect telemetry based on interval.
- bool is_telemetry_flush_time (uint32_t &collection_counter)

  Check if it's time to flush telemetry buffer based on count.
- uint32_t get_telemetry_sample_interval ()

  Get the current sample interval in milliseconds.
- void set_telemetry_sample_interval (uint32_t interval_ms)

  Set the telemetry sample interval.
- uint32_t get_telemetry_flush_threshold ()

  Get the number of records before flushing to storage.
- void set_telemetry_flush_threshold (uint32_t records)

  Set the number of records before flushing to storage.
- std::string get_last_telemetry_record_csv ()

  Gets the last telemetry record as a CSV string.
- std::string get_last_sensor_record_csv ()

  Gets the last sensor data record as a CSV string.

Variables

- constexpr int TELEMETRY_BUFFER_SIZE = 20

  Circular buffer for telemetry records.
- TelemetryRecord telemetry_buffer [TELEMETRY_BUFFER_SIZE]

  Circular buffer for telemetry records.
- size_t telemetry_buffer_count
- size_t telemetry_buffer_write_index
- SensorDataRecord sensor_data_buffer [TELEMETRY_BUFFER_SIZE]

  Circular buffer for sensor data records.
- size_t sensor_data_buffer_count
- size_t sensor_data_buffer_write_index
- mutex_t telemetry_mutex

  Mutex for thread-safe access to the telemetry buffer.

8.109.1 Detailed Description

System telemetry collection and logging.

This module handles periodic collection and storage of telemetry data from various satellite subsystems including power management, sensors (temperature, pressure, humidity, light), and GPS data.

Telemetry is collected at configurable intervals and stored in a circular buffer before being flushed to persistent storage after a configurable number of records are collected.

Definition in file telemetry_manager.h.

8.110 telemetry_manager.h

Go to the documentation of this file.

```
00001
00015
00016  #ifndef TELEMETRY_MANAGER_H
00017  #define TELEMETRY_MANAGER_H
00018
00019  #include <cstdint>
00020  #include <string>
00021  #include "pico/stdlib.h"
00022  #include "lib/location/NMEA/nmea_data.h"
00023  #include "utils.h"
00024  #include "storage.h"
00025  #include "PowerManager.h"
00026  #include "ISensor.h"
00027  #include "DS3231.h"
00028  #include <deque>
00029  #include <mutex>
00030  #include <iomanip>
00031  #include <sstream>
00032  #include <cstdio>
00033  #include "communication.h"
00034  #include <functional>
00035
00042 struct TelemetryRecord {
00043     uint32_t timestamp;
00044
00045     std::string build_version;
00046
00047     // Power data
00048     float battery_voltage;
00049     float system_voltage;
00050     float charge_current_usb;
00051     float charge_current_solar;
00052     float discharge_current;
```

```

    // GPS data - key RMC fields
    std::string time;
    std::string latitude;
    std::string lat_dir;
    std::string longitude;
    std::string lon_dir;
    std::string speed;
    std::string course;
    std::string date;

    // GPS data - key GGA fields
    std::string fix_quality;
    std::string satellites;
    std::string altitude;


    // Convert record to CSV line
    std::string to_csv() const {
        std::stringstream ss;
        ss « timestamp « ","
           « build_version « ","
           « std::fixed « std::setprecision(3)
           « battery_voltage « ","
           « system_voltage « ","
           « charge_current_usb « ","
           « charge_current_solar « ","
           « discharge_current « ","

           // GPS RMC data
           « time « ","
           « latitude « "," « lat_dir « ","
           « longitude « "," « lon_dir « ","
           « speed « ","
           « course « ","
           « date « ","
           // GPS GGA data
           « fix_quality « ","
           « satellites « ","
           « altitude;
        return ss.str();
    }
};


struct SensorDataRecord {
    uint32_t timestamp;
    float temperature;
    float pressure;
    float humidity;
    float light;

    // Convert record to CSV line
    std::string to_csv() const {
        std::stringstream ss;
        ss « timestamp « ","
           « std::fixed « std::setprecision(3)
           « temperature « ","
           « pressure « ","
           « humidity « ","
           « light;
        return ss.str();
    }
};

constexpr int TELEMETRY_BUFFER_SIZE = 20;
extern const int TELEMETRY_BUFFER_SIZE;
extern TelemetryRecord telemetry_buffer[TELEMETRY_BUFFER_SIZE];
extern size_t telemetry_buffer_count;
extern size_t telemetry_buffer_write_index;

extern SensorDataRecord sensor_data_buffer[TELEMETRY_BUFFER_SIZE];
extern size_t sensor_data_buffer_count;
extern size_t sensor_data_buffer_write_index;

extern mutex_t telemetry_mutex;


 bool telemetry_init();

 bool collect_telemetry();

 bool flush_telemetry();


bool flush_sensor_data();

```

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

bool is_telemetry_flush_time(uint32_t& collection_counter);

uint32_t get_telemetry_sample_interval();

void set_telemetry_sample_interval(uint32_t interval_ms);

uint32_t get_telemetry_flush_threshold();

void set_telemetry_flush_threshold(uint32_t records);


std::string get_last_telemetry_record_csv();

std::string get_last_sensor_record_csv();

#endif // TELEMETRY_MANAGER_H
 // End of TelemetryManager group
```

8.111 lib/utils.cpp File Reference

Implementation of utility functions for the Kubisat firmware.

```
#include "utils.h"
#include "pico/multicore.h"
#include "pico/sync.h"
#include <vector>
#include <queue>
#include <string>
#include <array>
```

Include dependency graph for utils.cpp:

Functions

- std::string get_level_color (VerbosityLevel level)

  Gets ANSI color code for verbosity level.
- std::string get_level_prefix (VerbosityLevel level)

  Gets text prefix for verbosity level.
- void uart_print (const std::string &msg, VerbosityLevel level, bool logToFile, uart_inst_t *uart)

  Prints a message to the UART with a timestamp and core number.
- std::string base64_encode (const uint8_t *input, size_t length)
- uint32_t calculate_checksum (const uint8_t *data, size_t length)

Variables

- static mutex_t uart_mutex

  Mutex for UART access protection.
- VerbosityLevel g_uart_verbosity = VerbosityLevel::DEBUG

  Global verbosity level setting.
- static const char base64_chars [ ]

8.111.1 Detailed Description

Implementation of utility functions for the Kubisat firmware.

Definition in file utils.cpp.

8.111.2 Function Documentation

8.111.2.1 get_level_color()

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

Gets ANSI color code for verbosity level.

Parameters

level	The verbosity level

Returns

ANSI color escape sequence

Definition at line 29 of file utils.cpp.

Here is the caller graph for this function:

8.111.2.2 get_level_prefix()

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

Gets text prefix for verbosity level.

Parameters

level	The verbosity level

Returns

Text prefix for the level

Definition at line 46 of file utils.cpp.

Here is the caller graph for this function:

8.111.2.3 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

msg	The message to print.
logToFile	A flag indicating whether to log the message to a file (currently not implemented).
uart	The UART instance to use for printing.

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

Definition at line 65 of file utils.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.111.2.4 base64_encode()

```
std::string base64_encode (
            const uint8_t * input,
            size_t length)
```

Definition at line 95 of file utils.cpp.

8.111.2.5 calculate_checksum()

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

Definition at line 139 of file utils.cpp.

8.111.3 Variable Documentation

8.111.3.1 uart_mutex

```
mutex_t uart_mutex  [static]
```

Mutex for UART access protection.

Definition at line 17 of file utils.cpp.

8.111.3.2 g_uart_verbosity

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

Global verbosity level setting.

Global verbosity level setting for UART output.

Definition at line 21 of file utils.cpp.

8.111.3.3 base64_chars

```
const char base64_chars[]  [static]
```

Initial value:

```
=
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"
```

Definition at line 92 of file utils.cpp.

8.112 utils.cpp

Go to the documentation of this file.

```
#include "utils.h"
#include "pico/multicore.h"
#include "pico/sync.h"
#include <vector>
#include <queue>
#include <string>
#include <array>



static mutex_t uart_mutex;


VerbosityLevel g_uart_verbosity = VerbosityLevel::DEBUG;


std::string get_level_color(VerbosityLevel level) {
    switch (level) {
        case VerbosityLevel::ERROR:   return ANSI_RED;
        case VerbosityLevel::WARNING: return ANSI_YELLOW;
        case VerbosityLevel::INFO:    return ANSI_GREEN;
        case VerbosityLevel::DEBUG:   return ANSI_BLUE;
        case VerbosityLevel::EVENT:    return ANSI_CYAN;
        default:                      return "";
    }
}


std::string get_level_prefix(VerbosityLevel level) {
    switch (level) {
        case VerbosityLevel::ERROR:   return "ERROR: ";
        case VerbosityLevel::WARNING: return "WARNING: ";
        case VerbosityLevel::INFO:    return "INFO: ";
        case VerbosityLevel::DEBUG:   return "DEBUG: ";
        case VerbosityLevel::EVENT:   return "EVENT: ";
        default:                      return "";
    }
}

void uart_print(const std::string& msg, VerbosityLevel level, bool logToFile, uart_inst_t* uart) {
    if (static_cast<int>(level) > static_cast<int>(g_uart_verbosity)) {
        return;
    }

    static bool mutex_inited = false;
    if (!mutex_inited) {
        mutex_init(&uart_mutex);
        mutex_inited = true;
    }

    uint32_t timestamp = to_ms_since_boot(get_absolute_time());
    uint core_num = get_core_num();

    std::string color = get_level_color(level);
    std::string prefix = get_level_prefix(level);
    std::string msg_to_send = "[" + std::to_string(timestamp) + "ms] - Core " +
                              std::to_string(core_num) + ": " +
                              color + prefix + ANSI_RESET + msg + "\r\n";

    mutex_enter_blocking(&uart_mutex);
    uart_puts(uart, msg_to_send.c_str());
    mutex_exit(&uart_mutex);
}


// Base64 encoding table
static const char base64_chars[] =
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

std::string base64_encode(const uint8_t* input, size_t length) {
    std::string result;
    result.reserve(((length + 2) / 3) * 4); // Reserve space for the result

    int i = 0;
    int j = 0;
    uint8_t char_array_3[3];
    uint8_t char_array_4[4];

    while (length--) {
        char_array_3[i++] = *(input++);
        if (i == 3) {
```

```
            char_array_4[0] = (char_array_3[0] & 0xfc) » 2;
            char_array_4[1] = ((char_array_3[0] & 0x03) « 4) + ((char_array_3[1] & 0xf0) » 4);
            char_array_4[2] = ((char_array_3[1] & 0x0f) « 2) + ((char_array_3[2] & 0xc0) » 6);
            char_array_4[3] = char_array_3[2] & 0x3f;

            for (i = 0; i < 4; i++)
                result += base64_chars[char_array_4[i]];
            i = 0;
        }
    }

    // Handle the remaining bytes
    if (i) {
        for (j = i; j < 3; j++)
            char_array_3[j] = '\0';

        char_array_4[0] = (char_array_3[0] & 0xfc) » 2;
        char_array_4[1] = ((char_array_3[0] & 0x03) « 4) + ((char_array_3[1] & 0xf0) » 4);
        char_array_4[2] = ((char_array_3[1] & 0x0f) « 2) + ((char_array_3[2] & 0xc0) » 6);

        for (j = 0; j < i + 1; j++)
            result += base64_chars[char_array_4[j]];

        while (i++ < 3)
            result += '='; // Padding
    }

    return result;
}



uint32_t calculate_checksum(const uint8_t* data, size_t length) {
    uint32_t checksum = 0;
    for (size_t i = 0; i < length; ++i) {
        checksum ^= data[i];
    }
    return checksum;
}
```

8.113 lib/utils.h File Reference

Utility functions and definitions for the Kubisat firmware.

```
#include <stdio.h>
#include <string>
#include "pico/stdlib.h"
#include "hardware/uart.h"
#include "pin_config.h"
#include <vector>
```

Include dependency graph for utils.h:

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

Macros

- #define ANSI_RED "\033[31m"

  ANSI escape codes for terminal color output.
- #define ANSI_GREEN "\033[32m"
- #define ANSI_YELLOW "\033[33m"
- #define ANSI_BLUE "\033[34m"
- #define ANSI_CYAN "\033[36m"
- #define ANSI_RESET "\033[0m"

Enumerations

- enum class VerbosityLevel {
  SILENT = 0 , ERROR = 1 , WARNING = 2 , INFO = 3 ,
  EVENT = 4 , DEBUG = 5 }

  Verbosity levels for logging system.

Functions

- void uart_print (const std::string &msg, VerbosityLevel level=VerbosityLevel::INFO, bool logToFile=false, uart_inst_t ∗uart=DEBUG_UART_PORT)

  Prints a message to UART with timestamp and formatting.
- std::string base64_encode (const uint8_t ∗input, size_t length)
- uint32_t calculate_checksum (const uint8_t ∗data, size_t length)

Variables

- VerbosityLevel g_uart_verbosity

  Global verbosity level setting for UART output.

8.113.1 Detailed Description

Utility functions and definitions for the Kubisat firmware.

Contains UART logging, color definitions, and CRC calculations

Definition in file utils.h.

8.113.2 Macro Definition Documentation

8.113.2.1 ANSI_RED

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

ANSI escape codes for terminal color output.

Definition at line 20 of file utils.h.

8.113.2.2 ANSI_GREEN

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

Definition at line 21 of file utils.h.

8.113.2.3 ANSI_YELLOW

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

Definition at line 22 of file utils.h.

8.113.2.4 ANSI_BLUE

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

Definition at line 23 of file utils.h.

8.113.2.5 ANSI_CYAN

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

Definition at line 24 of file utils.h.

8.113.2.6 ANSI_RESET

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

Definition at line 25 of file utils.h.

8.113.3 Enumeration Type Documentation

8.113.3.1 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.

8.113.4 Function Documentation

8.113.4.1 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

msg	The message to print
level	Message verbosity level
logToFile	Whether to store the message in log storage
uart	The UART port to use

Prints a message to UART with timestamp and formatting.

Parameters

msg	The message to print.
logToFile	A flag indicating whether to log the message to a file (currently not implemented).
uart	The UART instance to use for printing.

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

Definition at line 65 of file utils.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.113.4.2 base64_encode()

```
std::string base64_encode (
            const uint8_t * input,
            size_t length)
```

Definition at line 95 of file utils.cpp.

8.113.4.3 calculate_checksum()

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

Definition at line 139 of file utils.cpp.

8.113.5 Variable Documentation

8.113.5.1 g_uart_verbosity

```
VerbosityLevel g_uart_verbosity [extern]
```

Global verbosity level setting for UART output.

Controls which messages are displayed:

- SILENT (0): No output
- ERROR (1): Only errors
- WARNING (2): Warnings and errors
- INFO (3): Normal operation information
- DEBUG (4): Detailed debug information

Note

Can be changed at runtime through the command interface

See also

VerbosityLevel
handle_verbosity

Global verbosity level setting for UART output.

Definition at line 21 of file utils.cpp.

8.114 utils.h

Go to the documentation of this file.

```
#ifndef UTILS_H
#define UTILS_H

#include <stdio.h>
#include <string>
#include "pico/stdlib.h"
#include "hardware/uart.h"
#include "pin_config.h"
#include <vector>


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


enum class VerbosityLevel {
    SILENT = 0,
    ERROR = 1,
    WARNING = 2,
    INFO = 3,
    EVENT = 4,
    DEBUG = 5
};


extern VerbosityLevel g_uart_verbosity;


void uart_print(const std::string& msg,
               VerbosityLevel level = VerbosityLevel::INFO,
               bool logToFile = false,
               uart_inst_t* uart = DEBUG_UART_PORT);



std::string base64_encode(const uint8_t* input, size_t length);

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

8.115 main.cpp File Reference

`#include "includes.h"`

Include dependency graph for main.cpp:

Macros

- #define LOG_FILENAME "/log.txt"

Functions

- void core1_entry ()
- bool init_pico_hw ()
- bool init_modules ()
- int main ()

Variables

- PowerManager powerManager (MAIN_I2C_PORT)
- DS3231 systemClock (MAIN_I2C_PORT)
- volatile bool g_pending_bootloader_reset = false
- volatile bool pause_gps_collection = false
- char buffer [BUFFER_SIZE]
- int buffer_index = 0

8.115.1 Macro Definition Documentation

8.115.1.1 LOG_FILENAME

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

Definition at line 3 of file main.cpp.

8.115.2 Function Documentation

8.115.2.1 core1_entry()

```
void core1_entry ()
```

Definition at line 13 of file main.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.115.2.2 init_pico_hw()

```
bool init_pico_hw ()
```

Definition at line 58 of file main.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.115.2.3 init_modules()

```
bool init_modules ()
```

Definition at line 89 of file main.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.115.2.4 main()

```
int main (
            void )
```

Definition at line 146 of file main.cpp.

Here is the call graph for this function:

8.115.3 Variable Documentation

8.115.3.1 powerManager

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

8.115.3.2 systemClock

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

8.115.3.3 pause_gps_collection

```
volatile bool pause_gps_collection = false
```

Definition at line 8 of file main.cpp.

8.115.3.4 buffer

```
char buffer[BUFFER_SIZE]
```

Definition at line 10 of file main.cpp.

8.115.3.5 buffer_index

```
int buffer_index = 0
```

Definition at line 11 of file main.cpp.

8.116 main.cpp

Go to the documentation of this file.

```
00001 #include "includes.h"
00002
00003 #define LOG_FILENAME "/log.txt"
00004
00005 PowerManager powerManager(MAIN_I2C_PORT);
00006 DS3231 systemClock(MAIN_I2C_PORT);
00007 volatile bool g_pending_bootloader_reset = false;
00008 volatile bool pause_gps_collection = false;
00009
00010 char buffer[BUFFER_SIZE];
00011 int buffer_index = 0;
00012
00013 void core1_entry() {
00014     uart_print("Starting core 1", VerbosityLevel::DEBUG);
00015     EventEmitter::emit(EventGroup::SYSTEM, SystemEvent::CORE1_START);
00016
00017     uint32_t last_clock_check_time = 0;
00018     uint32_t last_telemetry_time = 0;
00019     uint32_t telemetry_collection_counter = 0;
00020
00021     telemetry_init();
00022
00023     while (true) {
00024         collect_gps_data();
00025
00026         uint32_t currentTime = to_ms_since_boot(get_absolute_time());
00027
00028         uint32_t check_interval_ms = systemClock.get_clock_sync_interval() * 60000;
00029         if (currentTime - last_clock_check_time >= check_interval_ms) {
00030             last_clock_check_time = currentTime;
00031
00032             if (systemClock.is_sync_needed()) {
00033                 uart_print("Clock sync interval reached, attempting sync", VerbosityLevel::INFO);
00034                 systemClock.sync_clock_with_gps();
00035             }
00036         }
```

```

        if (is_telemetry_collection_time(currentTime, last_telemetry_time)) {
            collect_telemetry();
            telemetry_collection_counter++;

            if (is_telemetry_flush_time(telemetry_collection_counter)) {
                flush_telemetry();
                flush_sensor_data();
            }
        }

        if (g_pending_bootloader_reset) {
            sleep_ms(100);
            uart_print("Entering BOOTSEL mode...", VerbosityLevel::WARNING);
            reset_usb_boot(0, 0);
        }

        sleep_ms(10);
    }
}

bool init_pico_hw() {
    stdio_init_all();

    uart_init(DEBUG_UART_PORT, DEBUG_UART_BAUD_RATE);
    gpio_set_function(DEBUG_UART_TX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_TX_PIN));
    gpio_set_function(DEBUG_UART_RX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_RX_PIN));

    uart_init(GPS_UART_PORT, GPS_UART_BAUD_RATE);
    gpio_set_function(GPS_UART_TX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_TX_PIN));
    gpio_set_function(GPS_UART_RX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_RX_PIN));

    gpio_init(PICO_DEFAULT_LED_PIN);
    gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);

    i2c_init(MAIN_I2C_PORT, 400 * 1000);
    gpio_set_function(MAIN_I2C_SCL_PIN, GPIO_FUNC_I2C);
    gpio_set_function(MAIN_I2C_SDA_PIN, GPIO_FUNC_I2C);
    gpio_pull_up(MAIN_I2C_SCL_PIN);
    gpio_pull_up(MAIN_I2C_SDA_PIN);

    gpio_init(GPS_POWER_ENABLE_PIN);
    gpio_set_dir(GPS_POWER_ENABLE_PIN, GPIO_OUT);
    gpio_put(GPS_POWER_ENABLE_PIN, 1);

    EventEmitter::emit(EventGroup::GPS, GPSEvent::POWER_ON);

    system("color");

    return true;
}

bool init_modules(){

    bool radio_init_status = false;
    radio_init_status = initialize_radio();

    bool sd_init_status = fs_init();
    if (sd_init_status) {
        FILE *fp = fopen(LOG_FILENAME, "w");
        if (fp) {
            uart_print("Log file opened.", VerbosityLevel::DEBUG);
            int bytes_written = fprintf(fp, "System init started.\n");
            uart_print("Written " + std::to_string(bytes_written) + " bytes.", VerbosityLevel::DEBUG);
            int close_status = fclose(fp);
            uart_print("Close file status: " + std::to_string(close_status), VerbosityLevel::DEBUG);

            struct stat file_stat;
            if (stat(LOG_FILENAME, &file_stat) == 0) {
                size_t file_size = file_stat.st_size;
                uart_print("File size: " + std::to_string(file_size) + " bytes",
    VerbosityLevel::DEBUG);
            } else {
                uart_print("Failed to get file size", VerbosityLevel::ERROR);
            }

            uart_print("File path: /" + std::string(LOG_FILENAME), VerbosityLevel::DEBUG);
        } else {
            uart_print("Failed to open log file for writing.", VerbosityLevel::ERROR);
        }
    }

    if (sd_init_status) {
        uart_print("SD card init: OK", VerbosityLevel::DEBUG);
    } else {
        uart_print("SD card init: FAILED", VerbosityLevel::ERROR);
    }
```

```

    if (radio_init_status) {
        uart_print("Radio init: OK", VerbosityLevel::DEBUG);
    } else {
        uart_print("Radio init: FAILED", VerbosityLevel::ERROR);
    }

    Frame boot = frame_build(OperationType::RES, 0, 0, "HELLO");
    send_frame_lora(boot);

    // uart_print("Initializing sensors...", VerbosityLevel::DEBUG);
    // SensorWrapper& sensor_wrapper = SensorWrapper::get_instance();
    // bool light_sensor_init = sensor_wrapper.sensor_init(SensorType::LIGHT, MAIN_I2C_PORT);
    // bool env_sensor_init = sensor_wrapper.sensor_init(SensorType::ENVIRONMENT, MAIN_I2C_PORT);
    // bool mag_sensor_init = sensor_wrapper.sensor_init(SensorType::MAGNETOMETER, MAIN_I2C_PORT);

    // if (!light_sensor_init || !env_sensor_init || !mag_sensor_init) {
    //     uart_print("One or more sensors failed to initialize", VerbosityLevel::WARNING);
    // }

    return sd_init_status && radio_init_status;
}

int main()
{
    init_pico_hw();
    init_modules();
    EventEmitter::emit(EventGroup::SYSTEM, SystemEvent::BOOT);
    multicore_launch_core1(core1_entry);

    gpio_put(PICO_DEFAULT_LED_PIN, 0);

    bool power_manager_init_status = powerManager.initialize();
    if (power_manager_init_status)
    {
        std::map<std::string, std::string> power_config = {
            {"operating_mode", "continuous"},
            {"averaging_mode", "16"},
        };
        powerManager.configure(power_config);
    } else {
        uart_print("Power manager init error", VerbosityLevel::ERROR);
    }

    Frame boot = frame_build(OperationType::RES, 0, 0, "START");
    send_frame_lora(boot);

    std::string boot_string = "System init completed @ " +
std::to_string(to_ms_since_boot(get_absolute_time())) + " ms";
    uart_print(boot_string, VerbosityLevel::WARNING);

    gpio_put(PICO_DEFAULT_LED_PIN, 1);

    while (true)
    {
        int packet_size = LoRa.parse_packet();
        if (packet_size)
        {
            on_receive(packet_size);
        }

        handle_uart_input();
    }

    return 0;
}
```

8.117 test/comms/test_comand_handlers.cpp File Reference

```
#include "unity.h"
#include "protocol.h"
#include "communication.h"
#include "commands.h"
```

Include dependency graph for test_comand_handlers.cpp:

Functions

- void send_frame_uart (const Frame &frame)
- void send_frame_lora (const Frame &frame)
- void setUp (void)
- void tearDown (void)
- void test_command_handler_get_operation (void)
- void test_command_handler_set_operation (void)
- void test_command_handler_invalid_operation (void)

Variables

- static bool uart_send_called = false
- static bool lora_send_called = false
- static Frame last_frame_sent

8.117.1 Function Documentation

8.117.1.1 send_frame_uart()

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

Definition at line 11 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.117.1.2 send_frame_lora()

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

Definition at line 16 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.117.1.3 setUp()

```
void setUp (
            void )
```

Definition at line 21 of file test_comand_handlers.cpp.

8.117.1.4 tearDown()

```
void tearDown (
            void )
```

Definition at line 26 of file test_comand_handlers.cpp.

8.117.1.5 test_command_handler_get_operation()

```
void test_command_handler_get_operation (
            void )
```

Definition at line 29 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.117.1.6 test_command_handler_set_operation()

```
void test_command_handler_set_operation (
            void )
```

Definition at line 39 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.117.1.7 test_command_handler_invalid_operation()

```
void test_command_handler_invalid_operation (
            void )
```

Definition at line 54 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.117.2 Variable Documentation

8.117.2.1 uart_send_called

```
bool uart_send_called = false  [static]
```

Definition at line 7 of file test_comand_handlers.cpp.

8.117.2.2 lora_send_called

```
bool lora_send_called = false  [static]
```

Definition at line 8 of file test_comand_handlers.cpp.

8.117.2.3 last_frame_sent

```
Frame last_frame_sent  [static]
```

Definition at line 9 of file test_comand_handlers.cpp.

8.118 test_comand_handlers.cpp

Go to the documentation of this file.

```
00001 // test/comms/test_command_handlers.cpp
00002 #include "unity.h"
00003 #include "protocol.h"
00004 #include "communication.h"
00005 #include "commands.h"
00006
00007 static bool uart_send_called = false;
00008 static bool lora_send_called = false;
00009 static Frame last_frame_sent;
00010
00011 void send_frame_uart(const Frame& frame) {
00012     uart_send_called = true;
00013     last_frame_sent = frame;
00014 }
00015
00016 void send_frame_lora(const Frame& frame) {
00017     lora_send_called = true;
00018     last_frame_sent = frame;
00019 }
00020
00021 void setUp(void) {
00022     uart_send_called = false;
00023     lora_send_called = false;
00024 }
00025
00026 void tearDown(void) {
00027 }
00028
00029 void test_command_handler_get_operation(void) {
00030     std::vector<Frame> response = handle_get_build_version("", OperationType::GET);
00031
00032     TEST_ASSERT_EQUAL(1, response.size());
00033     TEST_ASSERT_EQUAL(OperationType::VAL, response[0].operationType);
00034     TEST_ASSERT_EQUAL(1, response[0].group);
00035     TEST_ASSERT_EQUAL(1, response[0].command);
00036     TEST_ASSERT_EQUAL(BUILD_NUMBER, std::stoi(response[0].value));
00037 }
00038
00039 void test_command_handler_set_operation(void) {
00040     VerbosityLevel old_level = get_verbosity_level();
00041     std::vector<Frame> response = handle_verbosity("2", OperationType::SET);
00042
00043     TEST_ASSERT_EQUAL(1, response.size());
00044     TEST_ASSERT_EQUAL(OperationType::VAL, response[0].operationType);
00045     TEST_ASSERT_EQUAL(1, response[0].group);
00046     TEST_ASSERT_EQUAL(8, response[0].command);
00047     TEST_ASSERT_EQUAL_STRING("LEVEL SET", response[0].value.c_str());
00048
00049     TEST_ASSERT_EQUAL(VerbosityLevel::WARNING, get_verbosity_level());
00050
00051     set_verbosity_level(old_level);
00052 }
00053
00054 void test_command_handler_invalid_operation(void) {
00055     std::vector<Frame> response = handle_get_build_version("", OperationType::SET);
00056
00057     TEST_ASSERT_EQUAL(1, response.size());
00058     TEST_ASSERT_EQUAL(OperationType::ERR, response[0].operationType);
00059     TEST_ASSERT_EQUAL(1, response[0].group);
00060     TEST_ASSERT_EQUAL(1, response[0].command);
00061 }
```

8.119 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 ()

8.119.1 Function Documentation

8.119.1.1 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:

8.119.1.2 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:

8.119.1.3 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:

8.119.1.4 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:

8.120 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);
```

```
    OperationType converted = string_to_operation_type(str);

    TEST_ASSERT_EQUAL(type, converted);
    TEST_ASSERT_EQUAL_STRING("GET", str.c_str());
}

void test_value_unit_type_conversion() {
    ValueUnit unit = ValueUnit::VOLT;
    std::string str = value_unit_type_to_string(unit);

    TEST_ASSERT_EQUAL_STRING("V", str.c_str());
}

void test_exception_type_conversion() {
    ExceptionType type = ExceptionType::INVALID_PARAM;
    std::string str = exception_type_to_string(type);

    TEST_ASSERT_EQUAL_STRING("INVALID PARAM", str.c_str());
}

void test_hex_string_conversion() {
    std::string hex = "0A0B0C";
    std::vector<uint8_t> bytes = hex_string_to_bytes(hex);

    TEST_ASSERT_EQUAL(3, bytes.size());
    TEST_ASSERT_EQUAL(0x0A, bytes[0]);
    TEST_ASSERT_EQUAL(0x0B, bytes[1]);
    TEST_ASSERT_EQUAL(0x0C, bytes[2]);
}
```

8.121 test/comms/test_error_codes.cpp File Reference

```
#include "unity.h"
#include "protocol.h"
```

Include dependency graph for test_error_codes.cpp:

Functions

- void setUp (void)
- void tearDown (void)
- void test_error_code_conversion (void)

8.121.1 Function Documentation

8.121.1.1 setUp()

```
void setUp (
            void )
```

Definition at line 5 of file test_error_codes.cpp.

8.121.1.2 tearDown()

```
void tearDown (
            void )
```

Definition at line 6 of file test_error_codes.cpp.

8.121.1.3 test_error_code_conversion()

```
void test_error_code_conversion (
            void )
```

Definition at line 8 of file test_error_codes.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.122 test_error_codes.cpp

Go to the documentation of this file.

```
00001 // test/comms/test_error_codes.cpp
00002 #include "unity.h"
00003 #include "protocol.h"
00004
00005 void setUp(void) {}
00006 void tearDown(void) {}
00007
00008 void test_error_code_conversion(void) {
00009     // Test error code to string conversion
00010     TEST_ASSERT_EQUAL_STRING("PARAM_UNNECESSARY",
      error_code_to_string(ErrorCode::PARAM_UNNECESSARY).c_str());
00011     TEST_ASSERT_EQUAL_STRING("PARAM_REQUIRED",
      error_code_to_string(ErrorCode::PARAM_REQUIRED).c_str());
00012     TEST_ASSERT_EQUAL_STRING("PARAM_INVALID", error_code_to_string(ErrorCode::PARAM_INVALID).c_str());
00013     TEST_ASSERT_EQUAL_STRING("INVALID_OPERATION",
      error_code_to_string(ErrorCode::INVALID_OPERATION).c_str());
00014     TEST_ASSERT_EQUAL_STRING("NOT_ALLOWED", error_code_to_string(ErrorCode::NOT_ALLOWED).c_str());
00015     TEST_ASSERT_EQUAL_STRING("INVALID_FORMAT",
      error_code_to_string(ErrorCode::INVALID_FORMAT).c_str());
00016     TEST_ASSERT_EQUAL_STRING("INVALID_VALUE", error_code_to_string(ErrorCode::INVALID_VALUE).c_str());
00017     TEST_ASSERT_EQUAL_STRING("UNKNOWN_ERROR", error_code_to_string(ErrorCode::UNKNOWN_ERROR).c_str());
00018 }
```

8.123 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 ()

8.123.1 Function Documentation

8.123.1.1 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:

8.123.1.2 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:

8.123.1.3 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:

8.124 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 }
```

8.125 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 ()

8.125.1 Function Documentation

8.125.1.1 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:

8.125.1.2 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:

8.125.1.3 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:

8.126 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() {
```

```
    Frame frame = create_test_frame();
    std::string encoded = frame_encode(frame);

    TEST_ASSERT_NOT_EQUAL(0, encoded.length());
    TEST_ASSERT_TRUE(encoded.find(FRAME_BEGIN) != std::string::npos);
    TEST_ASSERT_TRUE(encoded.find(FRAME_END) != std::string::npos);
    TEST_ASSERT_TRUE(encoded.find("test_value") != std::string::npos);
}

void test_frame_decode_basic() {
    Frame original = create_test_frame();
    std::string encoded = frame_encode(original);
    Frame decoded = frame_decode(encoded);

    TEST_ASSERT_EQUAL(original.direction, decoded.direction);
    TEST_ASSERT_EQUAL(original.group, decoded.group);
    TEST_ASSERT_EQUAL(original.command, decoded.command);
    TEST_ASSERT_EQUAL_STRING(original.value.c_str(), decoded.value.c_str());
    TEST_ASSERT_EQUAL_STRING(original.unit.c_str(), decoded.unit.c_str());
}

void test_frame_decode_invalid_header() {
    std::string invalid_frame = "INVALID" + std::string(1, DELIMITER) + "rest_of_frame";
    bool exceptionThrown = false;

    try {
        Frame decoded = frame_decode(invalid_frame);
    } catch (const std::runtime_error& e) {
        exceptionThrown = true;
    } catch (...) {
        // Catch any other exceptions to avoid crashing the test
    }

    TEST_ASSERT_TRUE(exceptionThrown);
}
```

8.127 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 ()

8.127.1 Function Documentation

8.127.1.1 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:

8.128 test_frame_common.h

Go to the documentation of this file.

```
// test_frame_common.h
#ifndef TEST_FRAME_COMMON_H
#define TEST_FRAME_COMMON_H

#include "unity.h"
#include "protocol.h"
#include "communication.h"

// Helper function to create a test frame
Frame create_test_frame() {
    Frame frame;
    frame.header = FRAME_BEGIN;
    frame.direction = 1;
    frame.operationType = OperationType::GET;
    frame.group = 1;
    frame.command = 2;
    frame.value = "test_value";
    frame.unit = "V";
    frame.footer = FRAME_END;
    return frame;
}

#endif
```

8.129 test/comms/test_frame_send.cpp File Reference

```
#include "unity.h"
#include "communication.h"
#include "../mocks/hardware_mocks.h"
```

Include dependency graph for test_frame_send.cpp:

Functions

- void setUp (void)
- void tearDown (void)
- void test_send_frame_uart (void)

8.129.1 Function Documentation

8.129.1.1 setUp()

```
void setUp (
            void )
```

Definition at line 6 of file test_frame_send.cpp.

8.129.1.2 tearDown()

```
void tearDown (
            void )
```

Definition at line 12 of file test_frame_send.cpp.

8.129.1.3 test_send_frame_uart()

```
void test_send_frame_uart (
            void )
```

Definition at line 17 of file test_frame_send.cpp.

Here is the call graph for this function:

8.130 test_frame_send.cpp

Go to the documentation of this file.

```
// test/comms/test_frame_send.cpp
#include "unity.h"
#include "communication.h"
#include "../mocks/hardware_mocks.h"

void setUp(void) {
    // Enable mocks before each test
    mock_uart_enabled = true;
    uart_output_buffer.clear();
}

void tearDown(void) {
    // Disable mocks after each test
    mock_uart_enabled = false;
}

void test_send_frame_uart(void) {
    // Create a test frame
    Frame test_frame = {
        .operationType = OperationType::VAL,
        .group = 1,
        .command = 2,
        .value = "TEST_VALUE"
    };

    // Call function under test
    send_frame_uart(test_frame);

    // Verify output using mocks
    TEST_ASSERT_EQUAL(1, uart_output_buffer.size());
    TEST_ASSERT_TRUE(uart_output_buffer[0].find("KBST;0;VAL;1;2;TEST_VALUE;") != std::string::npos);
}
```

8.131 test/mocks/hardware_mocks.cpp File Reference

```
#include "hardware_mocks.h"
#include <cstring>
```

Include dependency graph for hardware_mocks.cpp:

Functions

- void mock_uart_puts (uart_inst_t ∗uart, const char ∗str)
- void mock_uart_init (uart_inst_t ∗uart, uint baudrate)
- void mock_spi_write_blocking (spi_inst_t ∗spi, const uint8_t ∗src, size_t len)
- int mock_spi_read_blocking (spi_inst_t ∗spi, uint8_t tx_data, uint8_t ∗dst, size_t len)

Variables

- bool mock_uart_enabled = false
- std::vector< std::string > uart_output_buffer
- bool mock_spi_enabled = false
- std::vector< uint8_t > spi_output_buffer

8.131.1 Function Documentation

8.131.1.1 mock_uart_puts()

```
void mock_uart_puts (
            uart_inst_t * uart,
            const char * str)
```

Definition at line 9 of file hardware_mocks.cpp.

8.131.1.2 mock_uart_init()

```
void mock_uart_init (
            uart_inst_t * uart,
            uint baudrate)
```

Definition at line 17 of file hardware_mocks.cpp.

8.131.1.3 mock_spi_write_blocking()

```
void mock_spi_write_blocking (
            spi_inst_t * spi,
            const uint8_t * src,
            size_t len)
```

Definition at line 27 of file hardware_mocks.cpp.

8.131.1.4 mock_spi_read_blocking()

```
int mock_spi_read_blocking (
            spi_inst_t * spi,
            uint8_t tx_data,
            uint8_t * dst,
            size_t len)
```

Definition at line 35 of file hardware_mocks.cpp.

8.131.2 Variable Documentation

8.131.2.1 mock_uart_enabled

```
bool mock_uart_enabled = false
```

Definition at line 6 of file hardware_mocks.cpp.

8.131.2.2 uart_output_buffer

```
std::vector<std::string> uart_output_buffer
```

Definition at line 7 of file hardware_mocks.cpp.

8.131.2.3 mock_spi_enabled

```
bool mock_spi_enabled = false
```

Definition at line 24 of file hardware_mocks.cpp.

8.131.2.4 spi_output_buffer

```
std::vector<uint8_t> spi_output_buffer
```

Definition at line 25 of file hardware_mocks.cpp.

8.132 hardware_mocks.cpp

Go to the documentation of this file.

```
00001 // test/mocks/hardware_mocks.cpp
00002 #include "hardware_mocks.h"
00003 #include <cstring>
00004
00005 // UART mocks
00006 bool mock_uart_enabled = false;
00007 std::vector<std::string> uart_output_buffer;
00008
00009 void mock_uart_puts(uart_inst_t* uart, const char* str) {
00010     if (mock_uart_enabled) {
00011         uart_output_buffer.push_back(std::string(str));
00012     } else {
00013         uart_puts(uart, str);
00014     }
00015 }
00016
00017 void mock_uart_init(uart_inst_t* uart, uint baudrate) {
00018     if (!mock_uart_enabled) {
00019         uart_init(uart, baudrate);
00020     }
00021 }
00022
00023 // SPI mocks
00024 bool mock_spi_enabled = false;
00025 std::vector<uint8_t> spi_output_buffer;
00026
00027 void mock_spi_write_blocking(spi_inst_t* spi, const uint8_t* src, size_t len) {
00028     if (mock_spi_enabled) {
00029         spi_output_buffer.insert(spi_output_buffer.end(), src, src + len);
00030     } else {
00031         spi_write_blocking(spi, src, len);
00032     }
00033 }
00034
00035 int mock_spi_read_blocking(spi_inst_t* spi, uint8_t tx_data, uint8_t* dst, size_t len) {
00036     if (mock_spi_enabled) {
00037         // Mock implementation that fills dst with test data
00038         memset(dst, tx_data, len);
00039         return len;
00040     } else {
00041         return spi_read_blocking(spi, tx_data, dst, len);
00042     }
00043 }
```

8.133 test/mocks/hardware_mocks.h File Reference

```
#include <vector>
#include <string>
```

Include dependency graph for hardware_mocks.h:

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

Functions

- void mock_uart_puts (uart_inst_t ∗uart, const char ∗str)
- void mock_uart_init (uart_inst_t ∗uart, uint baudrate)
- void mock_spi_write_blocking (spi_inst_t ∗spi, const uint8_t ∗src, size_t len)
- int mock_spi_read_blocking (spi_inst_t ∗spi, uint8_t tx_data, uint8_t ∗dst, size_t len)

Variables

- bool mock_uart_enabled
- std::vector< std::string > uart_output_buffer
- bool mock_spi_enabled
- std::vector< uint8_t > spi_output_buffer

8.133.1 Function Documentation

8.133.1.1 mock_uart_puts()

```
void mock_uart_puts (
            uart_inst_t * uart,
            const char * str)
```

Definition at line 9 of file hardware_mocks.cpp.

8.133.1.2 mock_uart_init()

```
void mock_uart_init (
            uart_inst_t * uart,
            uint baudrate)
```

Definition at line 17 of file hardware_mocks.cpp.

8.133.1.3 mock_spi_write_blocking()

```
void mock_spi_write_blocking (
            spi_inst_t * spi,
            const uint8_t * src,
            size_t len)
```

Definition at line 27 of file hardware_mocks.cpp.

8.133.1.4 mock_spi_read_blocking()

```
int mock_spi_read_blocking (
            spi_inst_t * spi,
            uint8_t tx_data,
            uint8_t * dst,
            size_t len)
```

Definition at line 35 of file hardware_mocks.cpp.

8.133.2 Variable Documentation

8.133.2.1 mock_uart_enabled

```
bool mock_uart_enabled  [extern]
```

Definition at line 6 of file hardware_mocks.cpp.

8.133.2.2 uart_output_buffer

```
std::vector<std::string> uart_output_buffer  [extern]
```

Definition at line 7 of file hardware_mocks.cpp.

8.133.2.3 mock_spi_enabled

```
bool mock_spi_enabled [extern]
```

Definition at line 24 of file hardware_mocks.cpp.

8.133.2.4 spi_output_buffer

```
std::vector<uint8_t> spi_output_buffer [extern]
```

Definition at line 25 of file hardware_mocks.cpp.

8.134 hardware_mocks.h

Go to the documentation of this file.

```
00001 // test/mocks/hardware_mocks.h
00002 #ifndef HARDWARE_MOCKS_H
00003 #define HARDWARE_MOCKS_H
00004
00005 #include <vector>
00006 #include <string>
00007
00008 // UART mocks
00009 extern bool mock_uart_enabled;
00010 extern std::vector<std::string> uart_output_buffer;
00011
00012 void mock_uart_puts(uart_inst_t* uart, const char* str);
00013 void mock_uart_init(uart_inst_t* uart, uint baudrate);
00014
00015 // SPI mocks
00016 extern bool mock_spi_enabled;
00017 extern std::vector<uint8_t> spi_output_buffer;
00018
00019 void mock_spi_write_blocking(spi_inst_t* spi, const uint8_t* src, size_t len);
00020 int mock_spi_read_blocking(spi_inst_t* spi, uint8_t tx_data, uint8_t* dst, size_t len);
00021
00022 #endif // HARDWARE_MOCKS_H
```

8.135 test/test_runner.cpp File Reference

```
#include "includes.h"
#include "unity.h"
```

Include dependency graph for test_runner.cpp:

Functions

- void test_frame_encode_basic (void)
- void test_frame_decode_basic (void)
- void test_frame_decode_invalid_header (void)
- void test_frame_build_success (void)
- void test_frame_build_error (void)
- void test_frame_build_info (void)
- void test_operation_type_conversion (void)
- void test_value_unit_type_conversion (void)
- void test_exception_type_conversion (void)
- void test_hex_string_conversion (void)
- void test_command_handler_get_operation (void)
- void test_command_handler_set_operation (void)
- void test_command_handler_invalid_operation (void)
- void test_error_code_conversion (void)
- int main (void)

8.135.1 Function Documentation

8.135.1.1 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:

8.135.1.2 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:

8.135.1.3 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:

8.135.1.4 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:

8.135.1.5 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:

8.135.1.6 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:

8.135.1.7 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:

8.135.1.8 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:

8.135.1.9 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:

8.135.1.10 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:

8.135.1.11 test_command_handler_get_operation()

```
void test_command_handler_get_operation (
            void )  [extern]
```

Definition at line 29 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.135.1.12 test_command_handler_set_operation()

```
void test_command_handler_set_operation (
            void )  [extern]
```

Definition at line 39 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.135.1.13 test_command_handler_invalid_operation()

```
void test_command_handler_invalid_operation (
            void )  [extern]
```

Definition at line 54 of file test_comand_handlers.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.135.1.14 test_error_code_conversion()

```
void test_error_code_conversion (
            void )  [extern]
```

Definition at line 8 of file test_error_codes.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.135.1.15 main()

```
int main (
            void )
```

Definition at line 26 of file test_runner.cpp.

Here is the call graph for this function:

8.136 test_runner.cpp

Go to the documentation of this file.

```
00001 // test/test_runner.cpp
```

```
#include "includes.h"
#include "unity.h"

// External test function declarations
// Pure software tests (no hardware dependencies)
extern void test_frame_encode_basic(void);
extern void test_frame_decode_basic(void);
extern void test_frame_decode_invalid_header(void);
extern void test_frame_build_success(void);
extern void test_frame_build_error(void);
extern void test_frame_build_info(void);
extern void test_operation_type_conversion(void);
extern void test_value_unit_type_conversion(void);
extern void test_exception_type_conversion(void);
extern void test_hex_string_conversion(void);

// Command handler tests
extern void test_command_handler_get_operation(void);
extern void test_command_handler_set_operation(void);
extern void test_command_handler_invalid_operation(void);

// Error code tests
extern void test_error_code_conversion(void);

int main(void) {
    stdio_init_all();
    uart_init(uart0, 115200);
    gpio_set_function(0, GPIO_FUNC_UART);
    gpio_set_function(1, GPIO_FUNC_UART);

    UNITY_BEGIN();
    uart_puts(uart0, "begin unity tests\n");

    // Frame codec tests (pure software)
    uart_puts(uart0, "begin frame codec tests\n");
    RUN_TEST(test_frame_encode_basic);
    RUN_TEST(test_frame_decode_basic);
    RUN_TEST(test_frame_decode_invalid_header);
    uart_puts(uart0, "end frame codec tests\n");

    // Frame build tests (pure software)
    uart_puts(uart0, "begin frame build tests\n");
    RUN_TEST(test_frame_build_success);
    RUN_TEST(test_frame_build_error);
    RUN_TEST(test_frame_build_info);
    uart_puts(uart0, "end frame build tests\n");

    // Converter tests (pure software)
    uart_puts(uart0, "begin converter tests\n");
    RUN_TEST(test_operation_type_conversion);
    RUN_TEST(test_value_unit_type_conversion);
    RUN_TEST(test_exception_type_conversion);
    RUN_TEST(test_hex_string_conversion);
    RUN_TEST(test_error_code_conversion);
    uart_puts(uart0, "end converter tests\n");

    // Command handler tests (pure software)
    uart_puts(uart0, "begin command handler tests\n");
    RUN_TEST(test_command_handler_get_operation);
    RUN_TEST(test_command_handler_set_operation);
    RUN_TEST(test_command_handler_invalid_operation);
    uart_puts(uart0, "end command handler tests\n");

    return UNITY_END();
}
```

Index

_BH1750_DEFAULT_MTREG
BH1750.h, 304
_BH1750_DEVICE_ID
BH1750.h, 303
_BH1750_MTREG_MAX
BH1750.h, 303
_BH1750_MTREG_MIN
BH1750.h, 303
__attribute__
Event Manager, 50, 52
_filterRes
INA3221, 152
_i2c
INA3221, 152
_i2c_addr
BH1750, 84
INA3221, 152
_masken_reg
INA3221, 152
_read
INA3221, 148
_shuntRes
INA3221, 152
_write
INA3221, 149
~EventManager
EventManager, 130
~ISensor
ISensor, 153

ACCEL_X
ISensor.h, 324
ACCEL_Y
ISensor.h, 324
ACCEL_Z
ISensor.h, 324
ADDR_SDO_HIGH
BME280, 91
ADDR_SDO_LOW
BME280, 91
address
HMC5883L, 143
Alert Functions, 65
enable_alerts, 67
get_crit_alert, 68
get_power_valid_alert, 70
get_warn_alert, 68
set_alert_latch, 70
set_crit_alert_limit, 66
set_power_valid_limit, 67
set_warn_alert_limit, 66
altitude
TelemetryRecord, 179
ANSI_BLUE
utils.h, 355
ANSI_CYAN
utils.h, 355
ANSI_GREEN
utils.h, 355
ANSI_RED
utils.h, 355
ANSI_RESET
utils.h, 355
ANSI_YELLOW
utils.h, 355
avg_mode
INA3221::conf_reg_t, 104

base64_chars
utils.cpp, 351
base64_encode
utils.cpp, 350
utils.h, 358
battery_voltage
TelemetryRecord, 176
bcd_to_bin
DS3231, 121
begin
BH1750, 82
Configuration Functions, 54
BH1750, 81
_i2c_addr, 84
begin, 82
BH1750, 82
configure, 82
CONTINUOUS_HIGH_RES_MODE, 82
CONTINUOUS_HIGH_RES_MODE_2, 82
CONTINUOUS_LOW_RES_MODE, 82
get_light_level, 83
Mode, 81
ONE_TIME_HIGH_RES_MODE, 82
ONE_TIME_HIGH_RES_MODE_2, 82
ONE_TIME_LOW_RES_MODE, 82
POWER_ON, 82
RESET, 82
UNCONFIGURED_POWER_DOWN, 82
write8, 83
BH1750.h
_BH1750_DEFAULT_MTREG, 304
_BH1750_DEVICE_ID, 303

_BH1750_MTREG_MAX, 303
_BH1750_MTREG_MIN, 303
BH1750Wrapper, 84
BH1750Wrapper, 85
configure, 86
get_address, 86
get_i2c_addr, 85
get_type, 86
init, 85
initialized_, 86
is_initialized, 86
read_data, 85
sensor_, 86
bin_to_bcd
DS3231, 120
BME280, 87
ADDR_SDO_HIGH, 91
ADDR_SDO_LOW, 91
BME280, 89
calib_params, 91
configure_sensor, 90
convert_humidity, 90
convert_pressure, 90
convert_temperature, 89
device_addr, 91
get_calibration_parameters, 90
i2c_port, 91
init, 89
initialized_, 91
NUM_CALIB_PARAMS, 96
read_raw_all, 89
REG_CONFIG, 91
REG_CTRL_HUM, 92
REG_CTRL_MEAS, 92
REG_DIG_H1, 95
REG_DIG_H2, 95
REG_DIG_H3, 96
REG_DIG_H4, 96
REG_DIG_H5, 96
REG_DIG_H6, 96
REG_DIG_P1_LSB, 93
REG_DIG_P1_MSB, 93
REG_DIG_P2_LSB, 93
REG_DIG_P2_MSB, 93
REG_DIG_P3_LSB, 94
REG_DIG_P3_MSB, 94
REG_DIG_P4_LSB, 94
REG_DIG_P4_MSB, 94
REG_DIG_P5_LSB, 94
REG_DIG_P5_MSB, 94
REG_DIG_P6_LSB, 94
REG_DIG_P6_MSB, 94
REG_DIG_P7_LSB, 95
REG_DIG_P7_MSB, 95
REG_DIG_P8_LSB, 95
REG_DIG_P8_MSB, 95
REG_DIG_P9_LSB, 95
REG_DIG_P9_MSB, 95
REG_DIG_T1_LSB, 92
REG_DIG_T1_MSB, 92
REG_DIG_T2_LSB, 93
REG_DIG_T2_MSB, 93
REG_DIG_T3_LSB, 93
REG_DIG_T3_MSB, 93
REG_HUMIDITY_MSB, 92
REG_PRESSURE_MSB, 92
REG_RESET, 92
REG_TEMPERATURE_MSB, 92
reset, 89
t_fine, 91
BME280CalibParam, 96
dig_h1, 99
dig_h2, 99
dig_h3, 99
dig_h4, 99
dig_h5, 99
dig_h6, 99
dig_p1, 97
dig_p2, 98
dig_p3, 98
dig_p4, 98
dig_p5, 98
dig_p6, 98
dig_p7, 98
dig_p8, 98
dig_p9, 98
dig_t1, 97
dig_t2, 97
dig_t3, 97
BME280Wrapper, 100
BME280Wrapper, 101
configure, 102
get_address, 102
get_type, 101
init, 101
initialized_, 102
is_initialized, 101
read_data, 101
sensor_, 102
BOOL
protocol.h, 240
BOOT
Event Manager, 49
buffer
main.cpp, 364
buffer_index
main.cpp, 364
BUFFER_SIZE
pin_config.h, 281
BUILD_NUMBER
build_number.h, 181
build_number.h, 181
BUILD_NUMBER, 181
build_version
TelemetryRecord, 176
bus_conv_time

INA3221::conf_reg_t, 103

calculate_checksum
utils.cpp, 350
utils.h, 358
calib_params
BME280, 91
century
ds3231_data_t, 125
ch1_en
INA3221::conf_reg_t, 104
ch2_en
INA3221::conf_reg_t, 104
ch3_en
INA3221::conf_reg_t, 104
CHANGED
Event Manager, 50
charge_current_solar
TelemetryRecord, 177
charge_current_usb
TelemetryRecord, 176
CHARGE_SOLAR
power_commands.cpp, 211
CHARGE_TOTAL
power_commands.cpp, 211
CHARGE_USB
power_commands.cpp, 210
charging_solar_active_
PowerManager, 167
charging_usb_active_
PowerManager, 167
check_power_alerts
PowerManager, 166
CLOCK
Event Manager, 48
Clock Commands, 1
Clock Management Commands, 11
handle_clock_sync_interval, 13
handle_get_last_sync_time, 14
handle_time, 11
handle_timezone_offset, 12
systemClock, 15
clock_commands.cpp
CLOCK_GROUP, 194
CLOCK_SYNC_INTERVAL, 194
LAST_SYNC_TIME, 194
TIME, 194
TIMEZONE_OFFSET, 194
clock_enable
DS3231, 112
CLOCK_GROUP
clock_commands.cpp, 194
clock_mutex_
DS3231, 122
CLOCK_SYNC_INTERVAL
clock_commands.cpp, 194
ClockEvent
Event Manager, 50
collect_gps_data
gps_collector.cpp, 269
gps_collector.h, 272
collect_telemetry
Telemetry Manager, 73
command
Frame, 139
Command System, 15
command_handlers, 17
CommandHandler, 15
CommandMap, 15
execute_command, 16
command_handlers
Command System, 17
CommandAccessLevel
protocol.h, 239
CommandHandler
Command System, 15
frame.cpp, 235
CommandMap
Command System, 15
COMMS
Event Manager, 48
CommsEvent
Event Manager, 49
communication.cpp
initialize_radio, 224
interval, 226
lastPrintTime, 226
lastReceiveTime, 226
lastSendTime, 226
lora_tx_done_callback, 225
msgCount, 226
outgoing, 226
communication.h
determine_unit, 233
handle_uart_input, 230
initialize_radio, 228
lora_tx_done_callback, 229
on_receive, 229
send_frame_lora, 232
send_frame_uart, 231
send_message, 231
split_and_send_message, 233
Configuration Functions, 53
begin, 54
get_die_id, 56
get_manufacturer_id, 55
INA3221, 54
read_register, 57
reset, 55
set_averaging_mode, 61
set_bus_conversion_time, 61
set_bus_measurement_disable, 60
set_bus_measurement_enable, 60
set_mode_continuous, 58
set_mode_power_down, 57
set_mode_triggered, 58
set_shunt_conversion_time, 62

set_shunt_measurement_disable, 59
set_shunt_measurement_enable, 59
configure
BH1750, 82
BH1750Wrapper, 86
BME280Wrapper, 102
HMC5883LWrapper, 145
ISensor, 153
MPU6050Wrapper, 158
PowerManager, 165
configure_sensor
BME280, 90
CONTINUOUS_HIGH_RES_MODE
BH1750, 82
CONTINUOUS_HIGH_RES_MODE_2
BH1750, 82
CONTINUOUS_LOW_RES_MODE
BH1750, 82
conv_ready
INA3221::masken_reg_t, 155
convert_humidity
BME280, 90
convert_pressure
BME280, 90
convert_temperature
BME280, 89
core1_entry
main.cpp, 361
CORE1_START
Event Manager, 49
CORE1_STOP
Event Manager, 49
course
TelemetryRecord, 178
create_test_frame
test_frame_common.h, 383
crit_alert_ch1
INA3221::masken_reg_t, 156
crit_alert_ch2
INA3221::masken_reg_t, 156
crit_alert_ch3
INA3221::masken_reg_t, 155
crit_alert_latch_en
INA3221::masken_reg_t, 156

DATA_READY
Event Manager, 50
date
ds3231_data_t, 125
TelemetryRecord, 178
DATETIME
protocol.h, 240
day
ds3231_data_t, 124
days_of_week
DS3231.h, 191
DEBUG
utils.h, 356
DEBUG_UART_BAUD_RATE
pin_config.h, 279
DEBUG_UART_PORT
pin_config.h, 279
DEBUG_UART_RX_PIN
pin_config.h, 280
DEBUG_UART_TX_PIN
pin_config.h, 280
DEFAULT_FLUSH_THRESHOLD
telemetry_manager.cpp, 336
DEFAULT_SAMPLE_INTERVAL_MS
telemetry_manager.cpp, 336
DELIMITER
protocol.h, 246
determine_unit
communication.h, 233
device_addr
BME280, 91
Diagnostic Commands, 17
g_pending_bootloader_reset, 21
handle_enter_bootloader_mode, 20
handle_get_build_version, 18
handle_get_commands_list, 18
handle_verbosity, 19
dig_h1
BME280CalibParam, 99
dig_h2
BME280CalibParam, 99
dig_h3
BME280CalibParam, 99
dig_h4
BME280CalibParam, 99
dig_h5
BME280CalibParam, 99
dig_h6
BME280CalibParam, 99
dig_p1
BME280CalibParam, 97
dig_p2
BME280CalibParam, 98
dig_p3
BME280CalibParam, 98
dig_p4
BME280CalibParam, 98
dig_p5
BME280CalibParam, 98
dig_p6
BME280CalibParam, 98
dig_p7
BME280CalibParam, 98
dig_p8
BME280CalibParam, 98
dig_p9
BME280CalibParam, 98
dig_t1
BME280CalibParam, 97
dig_t2
BME280CalibParam, 97
dig_t3

BME280CalibParam, 97
direction
Frame, 139
discharge_current
TelemetryRecord, 177
DRAW_TOTAL
power_commands.cpp, 211
DS3231, 104
bcd_to_bin, 121
bin_to_bcd, 120
clock_enable, 112
clock_mutex_, 122
DS3231, 106
ds3231_addr, 122
get_clock_sync_interval, 114
get_last_sync_time, 115
get_local_time, 116
get_time, 107
get_timezone_offset, 113
get_unix_time, 111
i2c, 122
i2c_read_reg, 118
i2c_write_reg, 119
is_sync_needed, 117
last_sync_time_, 123
read_temperature, 108
set_clock_sync_interval, 114
set_time, 106
set_timezone_offset, 113
set_unix_time, 110
sync_clock_with_gps, 117
sync_interval_minutes_, 123
timezone_offset_minutes_, 123
update_last_sync_time, 115
DS3231.h
days_of_week, 191
DS3231_CONTROL_REG, 190
DS3231_CONTROL_STATUS_REG, 191
DS3231_DATE_REG, 190
DS3231_DAY_REG, 190
DS3231_DEVICE_ADRESS, 189
DS3231_HOURS_REG, 190
DS3231_MINUTES_REG, 189
DS3231_MONTH_REG, 190
DS3231_SECONDS_REG, 189
DS3231_TEMPERATURE_LSB_REG, 191
DS3231_TEMPERATURE_MSB_REG, 191
DS3231_YEAR_REG, 190
FRIDAY, 191
MONDAY, 191
SATURDAY, 191
SUNDAY, 191
THURSDAY, 191
TUESDAY, 191
WEDNESDAY, 191
ds3231_addr
DS3231, 122
DS3231_CONTROL_REG
DS3231.h, 190
DS3231_CONTROL_STATUS_REG
DS3231.h, 191
ds3231_data_t, 123
century, 125
date, 125
day, 124
hours, 124
minutes, 124
month, 125
seconds, 124
year, 125
DS3231_DATE_REG
DS3231.h, 190
DS3231_DAY_REG
DS3231.h, 190
DS3231_DEVICE_ADRESS
DS3231.h, 189
DS3231_HOURS_REG
DS3231.h, 190
DS3231_MINUTES_REG
DS3231.h, 189
DS3231_MONTH_REG
DS3231.h, 190
DS3231_SECONDS_REG
DS3231.h, 189
DS3231_TEMPERATURE_LSB_REG
DS3231.h, 191
DS3231_TEMPERATURE_MSB_REG
DS3231.h, 191
DS3231_YEAR_REG
DS3231.h, 190

emit
EventEmitter, 126
enable_alerts
Alert Functions, 67
ENVIRONMENT
ISensor.h, 324
ERR
protocol.h, 239
ERROR
Event Manager, 50
utils.h, 356
error_code_to_string
protocol.h, 241
utils_converters.cpp, 257
ErrorCode
protocol.h, 238
EVENT
utils.h, 356
event
event_manager.h, 266
EventLog, 129
Event Commands, 21
handle_get_event_count, 23
handle_get_last_events, 22
Event Manager, 46
__attribute__, 50, 52

BOOT, 49
CHANGED, 50
CLOCK, 48
ClockEvent, 50
COMMS, 48
CommsEvent, 49
CORE1_START, 49
CORE1_STOP, 49
DATA_READY, 50
ERROR, 50
EventGroup, 48
eventLogId, 52
eventManager, 52
get_event, 51
GPS, 48
GPS_SYNC, 50
GPS_SYNC_DATA_NOT_READY, 50
GPSEvent, 49
LOCK, 50
log_event, 50
LOST, 50
LOW_BATTERY, 49
MSG_RECEIVED, 49
MSG_SENT, 49
OVERCHARGE, 49
PASS_THROUGH_END, 50
PASS_THROUGH_START, 50
POWER, 48
POWER_FALLING, 49
POWER_NORMAL, 49
POWER_OFF, 50
POWER_ON, 50
PowerEvent, 49
RADIO_ERROR, 49
RADIO_INIT, 49
SHUTDOWN, 49
SOLAR_ACTIVE, 49
SOLAR_INACTIVE, 49
SYSTEM, 48
systemClock, 52
SystemEvent, 48
UART_ERROR, 49
USB_CONNECTED, 49
USB_DISCONNECTED, 49
WATCHDOG_RESET, 49
EVENT_BUFFER_SIZE
event_manager.h, 265
EVENT_FLUSH_THRESHOLD
event_manager.h, 265
EVENT_LOG_FILE
event_manager.h, 265
event_manager.h
event, 266
EVENT_BUFFER_SIZE, 265
EVENT_FLUSH_THRESHOLD, 265
EVENT_LOG_FILE, 265
group, 266
id, 265
timestamp, 265
to_string, 265
eventCount
EventManager, 132
EventEmitter, 125
emit, 126
EventGroup
Event Manager, 48
EventLog, 127
event, 129
group, 128
id, 128
timestamp, 128
to_string, 128
eventLogId
Event Manager, 52
EventManager, 129
~EventManager, 130
eventCount, 132
EventManager, 130
eventMutex, 133
events, 132
eventsSinceFlush, 133
get_event_count, 131
init, 131
load_from_storage, 132
nextEventId, 133
save_to_storage, 131
writeIndex, 133
eventManager
Event Manager, 52
EventManagerImpl, 134
EventManagerImpl, 136
load_from_storage, 136
save_to_storage, 136
eventMutex
EventManager, 133
eventRegister
frame.cpp, 235
events
EventManager, 132
eventsSinceFlush
EventManager, 133
exception_type_to_string
protocol.h, 240
utils_converters.cpp, 254
ExceptionType
protocol.h, 240
execute_command
Command System, 16

FAIL_TO_SET
protocol.h, 239
FALL_RATE_THRESHOLD
PowerManager, 167
FALLING_TREND_REQUIRED
PowerManager, 167
fd
FileHandle, 137

FileHandle, 137
fd, 137
is_open, 137
fix_quality
TelemetryRecord, 178
flush_sensor_data
Telemetry Manager, 74
flush_telemetry
Telemetry Manager, 73
flush_threshold
telemetry_manager.cpp, 337
footer
Frame, 140
Frame, 138
command, 139
direction, 139
footer, 140
group, 139
header, 139
operationType, 139
unit, 140
value, 140
Frame Handling, 41
frame_build, 45
frame_decode, 43
frame_encode, 42
frame_process, 44
frame.cpp
CommandHandler, 235
eventRegister, 235
FRAME_BEGIN
protocol.h, 246
frame_build
Frame Handling, 45
frame_decode
Frame Handling, 43
frame_encode
Frame Handling, 42
FRAME_END
protocol.h, 246
frame_process
Frame Handling, 44
FRIDAY
DS3231.h, 191
fs_close_file
storage.h, 333
fs_file_exists
storage.h, 333
fs_init
storage.cpp, 329
storage.h, 331
fs_open_file
storage.h, 332
fs_read_file
storage.h, 332
fs_write_file
storage.h, 332

g_pending_bootloader_reset
Diagnostic Commands, 21
g_uart_verbosity
utils.cpp, 351
utils.h, 359
GET
protocol.h, 239
get_address
BH1750Wrapper, 86
BME280Wrapper, 102
HMC5883LWrapper, 145
ISensor, 154
get_available_sensors
SensorWrapper, 174
get_calibration_parameters
BME280, 90
get_clock_sync_interval
DS3231, 114
get_crit_alert
Alert Functions, 68
get_current
INA3221, 151
get_current_charge_solar
PowerManager, 164
get_current_charge_total
PowerManager, 164
get_current_charge_usb
PowerManager, 164
get_current_draw
PowerManager, 165
get_current_ma
Measurement Functions, 64
get_die_id
Configuration Functions, 56
get_event
Event Manager, 51
get_event_count
EventManager, 131
get_gga_tokens
NMEAData, 160
get_i2c_addr
BH1750Wrapper, 85
get_instance
SensorWrapper, 171
GET_LAST_SENSOR_RECORD_COMMAND
telemetry_commands.cpp, 222
get_last_sensor_record_csv
Telemetry Manager, 78
get_last_sync_time
DS3231, 115
GET_LAST_TELEMETRY_RECORD_COMMAND
telemetry_commands.cpp, 222
get_last_telemetry_record_csv
Telemetry Manager, 77
get_level_color
utils.cpp, 346
get_level_prefix
utils.cpp, 347
get_light_level

BH1750, 83
get_local_time
DS3231, 116
get_manufacturer_id
Configuration Functions, 55
get_power_valid_alert
Alert Functions, 70
get_rmc_tokens
NMEAData, 160
get_sensor
SensorWrapper, 173
get_shunt_voltage
Measurement Functions, 63
get_telemetry_flush_threshold
Telemetry Manager, 77
get_telemetry_sample_interval
Telemetry Manager, 76
get_time
DS3231, 107
get_timezone_offset
DS3231, 113
get_type
BH1750Wrapper, 86
BME280Wrapper, 101
HMC5883LWrapper, 145
ISensor, 153
MPU6050Wrapper, 158
get_unix_time
DS3231, 111
NMEAData, 161
get_voltage
Measurement Functions, 64
get_voltage_5v
PowerManager, 165
get_voltage_battery
PowerManager, 165
get_warn_alert
Alert Functions, 68
GGA_DATA_COMMAND
gps_commands.cpp, 206
gga_mutex_
NMEAData, 162
gga_tokens_
NMEAData, 161
GPS
Event Manager, 48
GPS Commands, 24
handle_enable_gps_uart_passthrough, 25
handle_get_gga_data, 27
handle_get_rmc_data, 26
handle_gps_power_status, 24
gps_collector.cpp
collect_gps_data, 269
MAX_RAW_DATA_LENGTH, 269
nmea_data, 270
pause_gps_collection, 270
splitString, 269
gps_collector.h
collect_gps_data, 272
gps_commands.cpp
GGA_DATA_COMMAND, 206
GPS_GROUP, 206
PASSTHROUGH_COMMAND, 206
POWER_STATUS_COMMAND, 206
RMC_DATA_COMMAND, 206
GPS_GROUP
gps_commands.cpp, 206
GPS_POWER_ENABLE_PIN
pin_config.h, 281
GPS_SYNC
Event Manager, 50
GPS_SYNC_DATA_NOT_READY
Event Manager, 50
GPS_UART_BAUD_RATE
pin_config.h, 280
GPS_UART_PORT
pin_config.h, 280
GPS_UART_RX_PIN
pin_config.h, 281
GPS_UART_TX_PIN
pin_config.h, 280
GPSEvent
Event Manager, 49
group
event_manager.h, 266
EventLog, 128
Frame, 139
GYRO_X
ISensor.h, 324
GYRO_Y
ISensor.h, 324
GYRO_Z
ISensor.h, 324

handle_clock_sync_interval
Clock Management Commands, 13
handle_enable_gps_uart_passthrough
GPS Commands, 25
handle_enter_bootloader_mode
Diagnostic Commands, 20
handle_get_build_version
Diagnostic Commands, 18
handle_get_commands_list
Diagnostic Commands, 18
handle_get_current_charge_solar
Power Commands, 31
handle_get_current_charge_total
Power Commands, 32
handle_get_current_charge_usb
Power Commands, 31
handle_get_current_draw
Power Commands, 33
handle_get_event_count
Event Commands, 23
handle_get_gga_data
GPS Commands, 27
handle_get_last_events

Event Commands, 22
handle_get_last_sensor_record
Telemetry Buffer Commands, 40
handle_get_last_sync_time
Clock Management Commands, 14
handle_get_last_telemetry_record
Telemetry Buffer Commands, 40
handle_get_power_manager_ids
Power Commands, 28
handle_get_rmc_data
GPS Commands, 26
handle_get_sensor_data
Sensor Commands, 34
handle_get_sensor_list
Sensor Commands, 36
handle_get_voltage_5v
Power Commands, 30
handle_get_voltage_battery
Power Commands, 29
handle_gps_power_status
GPS Commands, 24
handle_list_files
Storage Commands, 38
handle_mount
Storage Commands, 38
handle_sensor_config
Sensor Commands, 35
handle_time
Clock Management Commands, 11
handle_timezone_offset
Clock Management Commands, 12
handle_uart_input
communication.h, 230
receive.cpp, 249
handle_verbosity
Diagnostic Commands, 19
hardware_mocks.cpp
mock_spi_enabled, 387
mock_spi_read_blocking, 386
mock_spi_write_blocking, 386
mock_uart_enabled, 387
mock_uart_init, 386
mock_uart_puts, 386
spi_output_buffer, 387
uart_output_buffer, 387
hardware_mocks.h
mock_spi_enabled, 389
mock_spi_read_blocking, 389
mock_spi_write_blocking, 389
mock_uart_enabled, 389
mock_uart_init, 389
mock_uart_puts, 389
spi_output_buffer, 390
uart_output_buffer, 389
has_valid_time
NMEAData, 160
header
Frame, 139
hex_string_to_bytes
protocol.h, 243
utils_converters.cpp, 259
HMC5883L, 140
address, 143
HMC5883L, 141
i2c, 143
init, 141
read, 141
read_register, 142
write_register, 142
HMC5883LWrapper, 143
configure, 145
get_address, 145
get_type, 145
HMC5883LWrapper, 144
init, 145
initialized_, 146
is_initialized, 145
read_data, 145
sensor_, 146
hours
ds3231_data_t, 124
HUMIDITY
ISensor.h, 324
humidity
SensorDataRecord, 169

i2c
DS3231, 122
HMC5883L, 143
i2c_port
BME280, 91
i2c_read_reg
DS3231, 118
i2c_write_reg
DS3231, 119
id
event_manager.h, 265
EventLog, 128
IMU
ISensor.h, 324
INA3221, 146
_filterRes, 152
_i2c, 152
_i2c_addr, 152
_masken_reg, 152
_read, 148
_shuntRes, 152
_write, 149
Configuration Functions, 54
get_current, 151
INA3221 Power Monitor, 53
INA3221.h
INA3221_ADDR40_GND, 292
INA3221_ADDR41_VCC, 292
INA3221_ADDR42_SDA, 292
INA3221_ADDR43_SCL, 292
ina3221_addr_t, 292

ina3221_avg_mode_t, 293
INA3221_CH1, 292
INA3221_CH2, 292
INA3221_CH3, 292
INA3221_CH_NUM, 294
ina3221_ch_t, 292
ina3221_conv_time_t, 293
INA3221_REG_CH1_BUSV, 293
INA3221_REG_CH1_CRIT_ALERT_LIM, 293
INA3221_REG_CH1_SHUNTV, 293
INA3221_REG_CH1_WARNING_ALERT_LIM, 293
INA3221_REG_CH2_BUSV, 293
INA3221_REG_CH2_CRIT_ALERT_LIM, 293
INA3221_REG_CH2_SHUNTV, 293
INA3221_REG_CH2_WARNING_ALERT_LIM, 293
INA3221_REG_CH3_BUSV, 293
INA3221_REG_CH3_CRIT_ALERT_LIM, 293
INA3221_REG_CH3_SHUNTV, 293
INA3221_REG_CH3_WARNING_ALERT_LIM, 293
INA3221_REG_CONF, 293
INA3221_REG_CONF_AVG_1, 294
INA3221_REG_CONF_AVG_1024, 294
INA3221_REG_CONF_AVG_128, 294
INA3221_REG_CONF_AVG_16, 294
INA3221_REG_CONF_AVG_256, 294
INA3221_REG_CONF_AVG_4, 294
INA3221_REG_CONF_AVG_512, 294
INA3221_REG_CONF_AVG_64, 294
INA3221_REG_CONF_CT_1100US, 293
INA3221_REG_CONF_CT_140US, 293
INA3221_REG_CONF_CT_204US, 293
INA3221_REG_CONF_CT_2116US, 293
INA3221_REG_CONF_CT_332US, 293
INA3221_REG_CONF_CT_4156US, 293
INA3221_REG_CONF_CT_588US, 293
INA3221_REG_CONF_CT_8244US, 293
INA3221_REG_DIE_ID, 293
INA3221_REG_MANUF_ID, 293
INA3221_REG_MASK_ENABLE, 293
INA3221_REG_PWR_VALID_HI_LIM, 293
INA3221_REG_PWR_VALID_LO_LIM, 293
INA3221_REG_SHUNTV_SUM, 293
INA3221_REG_SHUNTV_SUM_LIM, 293
ina3221_reg_t, 292
SHUNT_VOLTAGE_LSB_UV, 294
INA3221::conf_reg_t, 102
avg_mode, 104
bus_conv_time, 103
ch1_en, 104
ch2_en, 104
ch3_en, 104
mode_bus_en, 103
mode_continious_en, 103
mode_shunt_en, 103
reset, 104
shunt_conv_time, 103
INA3221::masken_reg_t, 154
conv_ready, 155
crit_alert_ch1, 156
crit_alert_ch2, 156
crit_alert_ch3, 155
crit_alert_latch_en, 156
pwr_valid_alert, 155
reserved, 156
shunt_sum_alert, 155
shunt_sum_en_ch1, 156
shunt_sum_en_ch2, 156
shunt_sum_en_ch3, 156
timing_ctrl_alert, 155
warn_alert_ch1, 155
warn_alert_ch2, 155
warn_alert_ch3, 155
warn_alert_latch_en, 156
ina3221_
PowerManager, 167
INA3221_ADDR40_GND
INA3221.h, 292
INA3221_ADDR41_VCC
INA3221.h, 292
INA3221_ADDR42_SDA
INA3221.h, 292
INA3221_ADDR43_SCL
INA3221.h, 292
ina3221_addr_t
INA3221.h, 292
ina3221_avg_mode_t
INA3221.h, 293
INA3221_CH1
INA3221.h, 292
INA3221_CH2
INA3221.h, 292
INA3221_CH3
INA3221.h, 292
INA3221_CH_NUM
INA3221.h, 294
ina3221_ch_t
INA3221.h, 292
ina3221_conv_time_t
INA3221.h, 293
INA3221_REG_CH1_BUSV
INA3221.h, 293
INA3221_REG_CH1_CRIT_ALERT_LIM
INA3221.h, 293
INA3221_REG_CH1_SHUNTV
INA3221.h, 293
INA3221_REG_CH1_WARNING_ALERT_LIM
INA3221.h, 293
INA3221_REG_CH2_BUSV
INA3221.h, 293
INA3221_REG_CH2_CRIT_ALERT_LIM
INA3221.h, 293
INA3221_REG_CH2_SHUNTV
INA3221.h, 293

INA3221_REG_CH2_WARNING_ALERT_LIM
INA3221.h, 293
INA3221_REG_CH3_BUSV
INA3221.h, 293
INA3221_REG_CH3_CRIT_ALERT_LIM
INA3221.h, 293
INA3221_REG_CH3_SHUNTV
INA3221.h, 293
INA3221_REG_CH3_WARNING_ALERT_LIM
INA3221.h, 293
INA3221_REG_CONF
INA3221.h, 293
INA3221_REG_CONF_AVG_1
INA3221.h, 294
INA3221_REG_CONF_AVG_1024
INA3221.h, 294
INA3221_REG_CONF_AVG_128
INA3221.h, 294
INA3221_REG_CONF_AVG_16
INA3221.h, 294
INA3221_REG_CONF_AVG_256
INA3221.h, 294
INA3221_REG_CONF_AVG_4
INA3221.h, 294
INA3221_REG_CONF_AVG_512
INA3221.h, 294
INA3221_REG_CONF_AVG_64
INA3221.h, 294
INA3221_REG_CONF_CT_1100US
INA3221.h, 293
INA3221_REG_CONF_CT_140US
INA3221.h, 293
INA3221_REG_CONF_CT_204US
INA3221.h, 293
INA3221_REG_CONF_CT_2116US
INA3221.h, 293
INA3221_REG_CONF_CT_332US
INA3221.h, 293
INA3221_REG_CONF_CT_4156US
INA3221.h, 293
INA3221_REG_CONF_CT_588US
INA3221.h, 293
INA3221_REG_CONF_CT_8244US
INA3221.h, 293
INA3221_REG_DIE_ID
INA3221.h, 293
INA3221_REG_MANUF_ID
INA3221.h, 293
INA3221_REG_MASK_ENABLE
INA3221.h, 293
INA3221_REG_PWR_VALID_HI_LIM
INA3221.h, 293
INA3221_REG_PWR_VALID_LO_LIM
INA3221.h, 293
INA3221_REG_SHUNTV_SUM
INA3221.h, 293
INA3221_REG_SHUNTV_SUM_LIM
INA3221.h, 293
ina3221_reg_t
INA3221.h, 292
includes.h, 182
INFO
utils.h, 356
init
BH1750Wrapper, 85
BME280, 89
BME280Wrapper, 101
EventManager, 131
HMC5883L, 141
HMC5883LWrapper, 145
ISensor, 153
MPU6050Wrapper, 158
init_modules
main.cpp, 362
init_pico_hw
main.cpp, 361
initialize
PowerManager, 164
initialize_radio
communication.cpp, 224
communication.h, 228
initialized_
BH1750Wrapper, 86
BME280, 91
BME280Wrapper, 102
HMC5883LWrapper, 146
MPU6050Wrapper, 159
PowerManager, 167
Interface
protocol.h, 240
INTERNAL_FAIL_TO_READ
protocol.h, 239
interval
communication.cpp, 226
INVALID_FORMAT
protocol.h, 239
INVALID_OPERATION
protocol.h, 239, 240
INVALID_PARAM
protocol.h, 240
INVALID_VALUE
protocol.h, 239
is_charging_solar
PowerManager, 165
is_charging_usb
PowerManager, 165
is_initialized
BH1750Wrapper, 86
BME280Wrapper, 101
HMC5883LWrapper, 145
ISensor, 153
MPU6050Wrapper, 158
is_open
FileHandle, 137
is_sync_needed
DS3231, 117

is_telemetry_collection_time
Telemetry Manager, 75
is_telemetry_flush_time
Telemetry Manager, 76
ISensor, 152
~ISensor, 153
configure, 153
get_address, 154
get_type, 153
init, 153
is_initialized, 153
read_data, 153
ISensor.h
ACCEL_X, 324
ACCEL_Y, 324
ACCEL_Z, 324
ENVIRONMENT, 324
GYRO_X, 324
GYRO_Y, 324
GYRO_Z, 324
HUMIDITY, 324
IMU, 324
LIGHT, 324
LIGHT_LEVEL, 324
MAG_FIELD_X, 324
MAG_FIELD_Y, 324
MAG_FIELD_Z, 324
MAGNETOMETER, 324
PRESSURE, 324
SensorDataTypeIdentifier, 324
SensorType, 324
TEMPERATURE, 324

last_frame_sent
test_comand_handlers.cpp, 371
LAST_SYNC_TIME
clock_commands.cpp, 194
last_sync_time_
DS3231, 123
lastPrintTime
communication.cpp, 226
lastReceiveTime
communication.cpp, 226
lastSendTime
communication.cpp, 226
lat_dir
TelemetryRecord, 177
latitude
TelemetryRecord, 177
lib/clock/DS3231.cpp, 183
lib/clock/DS3231.h, 188, 192
lib/comms/commands/clock_commands.cpp, 193, 195
lib/comms/commands/commands.cpp, 197, 198
lib/comms/commands/commands.h, 199, 201
lib/comms/commands/diagnostic_commands.cpp, 201, 202
lib/comms/commands/event_commands.cpp, 204
lib/comms/commands/gps_commands.cpp, 205, 207
lib/comms/commands/power_commands.cpp, 209, 211
lib/comms/commands/sensor_commands.cpp, 213, 215
lib/comms/commands/storage_commands.cpp, 218, 220
lib/comms/commands/telemetry_commands.cpp, 221, 223
lib/comms/communication.cpp, 224, 227
lib/comms/communication.h, 227, 233
lib/comms/frame.cpp, 234, 235
lib/comms/protocol.h, 237, 246
lib/comms/receive.cpp, 247, 250
lib/comms/send.cpp, 251, 253
lib/comms/utils_converters.cpp, 254, 260
lib/eventman/event_manager.cpp, 261, 262
lib/eventman/event_manager.h, 263, 266
lib/location/gps_collector.cpp, 268, 271
lib/location/gps_collector.h, 271, 273
lib/location/NMEA/NMEA_data.cpp, 273, 274
lib/location/NMEA/NMEA_data.h, 275, 276
lib/pin_config.cpp, 277, 278
lib/pin_config.h, 278, 285
lib/powerman/INA3221/INA3221.cpp, 285, 286
lib/powerman/INA3221/INA3221.h, 290, 295
lib/powerman/PowerManager.cpp, 297
lib/powerman/PowerManager.h, 299, 300
lib/sensors/BH1750/BH1750.cpp, 301
lib/sensors/BH1750/BH1750.h, 302, 304
lib/sensors/BH1750/BH1750_WRAPPER.cpp, 304, 305
lib/sensors/BH1750/BH1750_WRAPPER.h, 306, 307
lib/sensors/BME280/BME280.cpp, 307, 308
lib/sensors/BME280/BME280.h, 311, 312
lib/sensors/BME280/BME280_WRAPPER.cpp, 314
lib/sensors/BME280/BME280_WRAPPER.h, 315
lib/sensors/HMC5883L/HMC5883L.cpp, 316
lib/sensors/HMC5883L/HMC5883L.h, 317, 318
lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp, 318, 319
lib/sensors/HMC5883L/HMC5883L_WRAPPER.h, 320, 321
lib/sensors/ISensor.cpp, 321, 322
lib/sensors/ISensor.h, 323, 325
lib/sensors/MPU6050/MPU6050.cpp, 325
lib/sensors/MPU6050/MPU6050.h, 326
lib/sensors/MPU6050/MPU6050_WRAPPER.cpp, 326
lib/sensors/MPU6050/MPU6050_WRAPPER.h, 326, 327
lib/storage/storage.cpp, 328, 330
lib/storage/storage.h, 330, 333
lib/telemetry/telemetry_manager.cpp, 334, 337
lib/telemetry/telemetry_manager.h, 341, 343
lib/utils.cpp, 345, 352
lib/utils.h, 353, 359
LIGHT
ISensor.h, 324
light
SensorDataRecord, 169
LIGHT_LEVEL
ISensor.h, 324
LIST_FILES_COMMAND

storage_commands.cpp, 219
load_from_storage
EventManager, 132
EventManagerImpl, 136
LOCK
Event Manager, 50
log_event
Event Manager, 50
LOG_FILENAME
main.cpp, 361
lon_dir
TelemetryRecord, 178
longitude
TelemetryRecord, 177
LORA
protocol.h, 240
lora_address_local
pin_config.cpp, 277
pin_config.h, 284
lora_address_remote
pin_config.cpp, 278
pin_config.h, 284
lora_cs_pin
pin_config.cpp, 277
pin_config.h, 284
LORA_DEFAULT_DIO0_PIN
pin_config.h, 283
LORA_DEFAULT_RESET_PIN
pin_config.h, 283
LORA_DEFAULT_SPI
pin_config.h, 282
LORA_DEFAULT_SPI_FREQUENCY
pin_config.h, 283
LORA_DEFAULT_SS_PIN
pin_config.h, 283
lora_irq_pin
pin_config.cpp, 277
pin_config.h, 284
lora_reset_pin
pin_config.cpp, 277
pin_config.h, 284
lora_send_called
test_comand_handlers.cpp, 370
lora_tx_done_callback
communication.cpp, 225
communication.h, 229
LOST
Event Manager, 50
LOW_BATTERY
Event Manager, 49

MAG_FIELD_X
ISensor.h, 324
MAG_FIELD_Y
ISensor.h, 324
MAG_FIELD_Z
ISensor.h, 324
MAGNETOMETER
ISensor.h, 324
main
main.cpp, 363
test_runner.cpp, 400
main.cpp, 360
buffer, 364
buffer_index, 364
core1_entry, 361
init_modules, 362
init_pico_hw, 361
LOG_FILENAME, 361
main, 363
pause_gps_collection, 364
powerManager, 363
systemClock, 363
MAIN_I2C_PORT
pin_config.h, 280
MAIN_I2C_SCL_PIN
pin_config.h, 280
MAIN_I2C_SDA_PIN
pin_config.h, 280
MAX_RAW_DATA_LENGTH
gps_collector.cpp, 269
Measurement Functions, 63
get_current_ma, 64
get_shunt_voltage, 63
get_voltage, 64
MILIAMP
protocol.h, 240
minutes
ds3231_data_t, 124
mock_spi_enabled
hardware_mocks.cpp, 387
hardware_mocks.h, 389
mock_spi_read_blocking
hardware_mocks.cpp, 386
hardware_mocks.h, 389
mock_spi_write_blocking
hardware_mocks.cpp, 386
hardware_mocks.h, 389
mock_uart_enabled
hardware_mocks.cpp, 387
hardware_mocks.h, 389
mock_uart_init
hardware_mocks.cpp, 386
hardware_mocks.h, 389
mock_uart_puts
hardware_mocks.cpp, 386
hardware_mocks.h, 389
Mode
BH1750, 81
mode_bus_en
INA3221::conf_reg_t, 103
mode_continious_en
INA3221::conf_reg_t, 103
mode_shunt_en
INA3221::conf_reg_t, 103
MONDAY
DS3231.h, 191

month
ds3231_data_t, 125
MOUNT_COMMAND
storage_commands.cpp, 219
MPU6050Wrapper, 157
configure, 158
get_type, 158
init, 158
initialized_, 159
is_initialized, 158
MPU6050Wrapper, 158
read_data, 158
sensor_, 159
MSG_RECEIVED
Event Manager, 49
MSG_SENT
Event Manager, 49
msgCount
communication.cpp, 226

nextEventId
EventManager, 133
nmea_data
gps_collector.cpp, 270
NMEA_data.cpp, 274
NMEA_data.h, 276
telemetry_manager.cpp, 337
NMEA_data.cpp
nmea_data, 274
NMEA_data.h
nmea_data, 276
NMEAData, 159
get_gga_tokens, 160
get_rmc_tokens, 160
get_unix_time, 161
gga_mutex_, 162
gga_tokens_, 161
has_valid_time, 160
NMEAData, 160
rmc_mutex_, 161
rmc_tokens_, 161
update_gga_tokens, 160
update_rmc_tokens, 160
NONE
protocol.h, 239, 240
NOT_ALLOWED
protocol.h, 239, 240
NUM_CALIB_PARAMS
BME280, 96

on_receive
communication.h, 229
receive.cpp, 248
ONE_TIME_HIGH_RES_MODE
BH1750, 82
ONE_TIME_HIGH_RES_MODE_2
BH1750, 82
ONE_TIME_LOW_RES_MODE
BH1750, 82
operation_type_to_string
protocol.h, 242
utils_converters.cpp, 256
OperationType
protocol.h, 239
operationType
Frame, 139
outgoing
communication.cpp, 226
OVERCHARGE
Event Manager, 49

PA_OUTPUT_PA_BOOST_PIN
pin_config.h, 283
PA_OUTPUT_RFO_PIN
pin_config.h, 283
PARAM_INVALID
protocol.h, 239
PARAM_REQUIRED
protocol.h, 239
PARAM_UNECESSARY
protocol.h, 240
PARAM_UNNECESSARY
protocol.h, 239
PASS_THROUGH_END
Event Manager, 50
PASS_THROUGH_START
Event Manager, 50
PASSTHROUGH_COMMAND
gps_commands.cpp, 206
pause_gps_collection
gps_collector.cpp, 270
main.cpp, 364
pin_config.cpp
lora_address_local, 277
lora_address_remote, 278
lora_cs_pin, 277
lora_irq_pin, 277
lora_reset_pin, 277
pin_config.h
BUFFER_SIZE, 281
DEBUG_UART_BAUD_RATE, 279
DEBUG_UART_PORT, 279
DEBUG_UART_RX_PIN, 280
DEBUG_UART_TX_PIN, 280
GPS_POWER_ENABLE_PIN, 281
GPS_UART_BAUD_RATE, 280
GPS_UART_PORT, 280
GPS_UART_RX_PIN, 281
GPS_UART_TX_PIN, 280
lora_address_local, 284
lora_address_remote, 284
lora_cs_pin, 284
LORA_DEFAULT_DIO0_PIN, 283
LORA_DEFAULT_RESET_PIN, 283
LORA_DEFAULT_SPI, 282
LORA_DEFAULT_SPI_FREQUENCY, 283
LORA_DEFAULT_SS_PIN, 283
lora_irq_pin, 284

lora_reset_pin, 284
MAIN_I2C_PORT, 280
MAIN_I2C_SCL_PIN, 280
MAIN_I2C_SDA_PIN, 280
PA_OUTPUT_PA_BOOST_PIN, 283
PA_OUTPUT_RFO_PIN, 283
READ_BIT, 282
SD_CARD_DETECT_PIN, 282
SD_CS_PIN, 281
SD_MISO_PIN, 281
SD_MOSI_PIN, 281
SD_SCK_PIN, 281
SD_SPI_PORT, 281
SPI_PORT, 282
SX1278_CS, 282
SX1278_MISO, 282
SX1278_MOSI, 282
SX1278_SCK, 282
POWER
Event Manager, 48
Power Commands, 28
handle_get_current_charge_solar, 31
handle_get_current_charge_total, 32
handle_get_current_charge_usb, 31
handle_get_current_draw, 33
handle_get_power_manager_ids, 28
handle_get_voltage_5v, 30
handle_get_voltage_battery, 29
power_commands.cpp
CHARGE_SOLAR, 211
CHARGE_TOTAL, 211
CHARGE_USB, 210
DRAW_TOTAL, 211
POWER_GROUP, 210
POWER_MANAGER_IDS, 210
VOLTAGE_BATTERY, 210
VOLTAGE_MAIN, 210
POWER_FALLING
Event Manager, 49
POWER_GROUP
power_commands.cpp, 210
POWER_MANAGER_IDS
power_commands.cpp, 210
POWER_NORMAL
Event Manager, 49
POWER_OFF
Event Manager, 50
POWER_ON
BH1750, 82
Event Manager, 50
POWER_STATUS_COMMAND
gps_commands.cpp, 206
PowerEvent
Event Manager, 49
powerman_mutex_
PowerManager, 167
PowerManager, 162
charging_solar_active_, 167
charging_usb_active_, 167
check_power_alerts, 166
configure, 165
FALL_RATE_THRESHOLD, 167
FALLING_TREND_REQUIRED, 167
get_current_charge_solar, 164
get_current_charge_total, 164
get_current_charge_usb, 164
get_current_draw, 165
get_voltage_5v, 165
get_voltage_battery, 165
ina3221_, 167
initialize, 164
initialized_, 167
is_charging_solar, 165
is_charging_usb, 165
powerman_mutex_, 167
PowerManager, 163
read_device_ids, 164
SOLAR_CURRENT_THRESHOLD, 166
USB_CURRENT_THRESHOLD, 166
VOLTAGE_LOW_THRESHOLD, 166
VOLTAGE_OVERCHARGE_THRESHOLD, 167
powerManager
main.cpp, 363
telemetry_manager.cpp, 336
PRESSURE
ISensor.h, 324
pressure
SensorDataRecord, 169
protocol.h
BOOL, 240
CommandAccessLevel, 239
DATETIME, 240
DELIMITER, 246
ERR, 239
error_code_to_string, 241
ErrorCode, 238
exception_type_to_string, 240
ExceptionType, 240
FAIL_TO_SET, 239
FRAME_BEGIN, 246
FRAME_END, 246
GET, 239
hex_string_to_bytes, 243
Interface, 240
INTERNAL_FAIL_TO_READ, 239
INVALID_FORMAT, 239
INVALID_OPERATION, 239, 240
INVALID_PARAM, 240
INVALID_VALUE, 239
LORA, 240
MILIAMP, 240
NONE, 239, 240
NOT_ALLOWED, 239, 240
operation_type_to_string, 242
OperationType, 239
PARAM_INVALID, 239

PARAM_REQUIRED, 239
PARAM_UNECESSARY, 240
PARAM_UNNECESSARY, 239
READ_ONLY, 239
READ_WRITE, 239
RES, 239
SECOND, 240
SEQ, 239
SET, 239
string_to_operation_type, 243
TEXT, 240
UART, 240
UNDEFINED, 240
UNKNOWN_ERROR, 239
VAL, 239
value_unit_type_to_string, 244
ValueUnit, 239
VOLT, 240
WRITE_ONLY, 239
pwr_valid_alert
INA3221::masken_reg_t, 155

RADIO_ERROR
Event Manager, 49
RADIO_INIT
Event Manager, 49
read
HMC5883L, 141
READ_BIT
pin_config.h, 282
read_data
BH1750Wrapper, 85
BME280Wrapper, 101
HMC5883LWrapper, 145
ISensor, 153
MPU6050Wrapper, 158
read_device_ids
PowerManager, 164
READ_ONLY
protocol.h, 239
read_raw_all
BME280, 89
read_register
Configuration Functions, 57
HMC5883L, 142
read_temperature
DS3231, 108
READ_WRITE
protocol.h, 239
receive.cpp
handle_uart_input, 249
on_receive, 248
REG_CONFIG
BME280, 91
REG_CTRL_HUM
BME280, 92
REG_CTRL_MEAS
BME280, 92
REG_DIG_H1
BME280, 95
REG_DIG_H2
BME280, 95
REG_DIG_H3
BME280, 96
REG_DIG_H4
BME280, 96
REG_DIG_H5
BME280, 96
REG_DIG_H6
BME280, 96
REG_DIG_P1_LSB
BME280, 93
REG_DIG_P1_MSB
BME280, 93
REG_DIG_P2_LSB
BME280, 93
REG_DIG_P2_MSB
BME280, 93
REG_DIG_P3_LSB
BME280, 94
REG_DIG_P3_MSB
BME280, 94
REG_DIG_P4_LSB
BME280, 94
REG_DIG_P4_MSB
BME280, 94
REG_DIG_P5_LSB
BME280, 94
REG_DIG_P5_MSB
BME280, 94
REG_DIG_P6_LSB
BME280, 94
REG_DIG_P6_MSB
BME280, 94
REG_DIG_P7_LSB
BME280, 95
REG_DIG_P7_MSB
BME280, 95
REG_DIG_P8_LSB
BME280, 95
REG_DIG_P8_MSB
BME280, 95
REG_DIG_P9_LSB
BME280, 95
REG_DIG_P9_MSB
BME280, 95
REG_DIG_T1_LSB
BME280, 92
REG_DIG_T1_MSB
BME280, 92
REG_DIG_T2_LSB
BME280, 93
REG_DIG_T2_MSB
BME280, 93
REG_DIG_T3_LSB
BME280, 93
REG_DIG_T3_MSB

BME280, 93
REG_HUMIDITY_MSB
BME280, 92
REG_PRESSURE_MSB
BME280, 92
REG_RESET
BME280, 92
REG_TEMPERATURE_MSB
BME280, 92
RES
protocol.h, 239
reserved
INA3221::masken_reg_t, 156
RESET
BH1750, 82
reset
BME280, 89
Configuration Functions, 55
INA3221::conf_reg_t, 104
RMC_DATA_COMMAND
gps_commands.cpp, 206
rmc_mutex_
NMEAData, 161
rmc_tokens_
NMEAData, 161

sample_interval_ms
telemetry_manager.cpp, 337
satellites
TelemetryRecord, 178
SATURDAY
DS3231.h, 191
save_to_storage
EventManager, 131
EventManagerImpl, 136
scan_connected_sensors
SensorWrapper, 174
SD_CARD_DETECT_PIN
pin_config.h, 282
sd_card_mounted
storage.cpp, 330
storage.h, 333
SD_CS_PIN
pin_config.h, 281
SD_MISO_PIN
pin_config.h, 281
SD_MOSI_PIN
pin_config.h, 281
SD_SCK_PIN
pin_config.h, 281
SD_SPI_PORT
pin_config.h, 281
SECOND
protocol.h, 240
seconds
ds3231_data_t, 124
send.cpp
send_frame_lora, 252
send_frame_uart, 252
send_message, 251
split_and_send_message, 252
send_frame_lora
communication.h, 232
send.cpp, 252
test_comand_handlers.cpp, 368
send_frame_uart
communication.h, 231
send.cpp, 252
test_comand_handlers.cpp, 367
send_message
communication.h, 231
send.cpp, 251
Sensor Commands, 34
handle_get_sensor_data, 34
handle_get_sensor_list, 36
handle_sensor_config, 35
sensor_
BH1750Wrapper, 86
BME280Wrapper, 102
HMC5883LWrapper, 146
MPU6050Wrapper, 159
sensor_commands.cpp
SENSOR_CONFIGURE, 214
SENSOR_GROUP, 214
SENSOR_READ, 214
SENSOR_CONFIGURE
sensor_commands.cpp, 214
sensor_configure
SensorWrapper, 172
sensor_data_buffer
Telemetry Manager, 80
sensor_data_buffer_count
Telemetry Manager, 80
sensor_data_buffer_write_index
Telemetry Manager, 80
SENSOR_DATA_CSV_PATH
telemetry_manager.cpp, 336
SENSOR_GROUP
sensor_commands.cpp, 214
sensor_init
SensorWrapper, 172
SENSOR_READ
sensor_commands.cpp, 214
sensor_read_data
SensorWrapper, 173
SensorDataRecord, 168
humidity, 169
light, 169
pressure, 169
temperature, 169
timestamp, 169
to_csv, 168
SensorDataTypeIdentifier
ISensor.h, 324
sensors
SensorWrapper, 175
SensorType

ISensor.h, 324
SensorWrapper, 170
get_available_sensors, 174
get_instance, 171
get_sensor, 173
scan_connected_sensors, 174
sensor_configure, 172
sensor_init, 172
sensor_read_data, 173
sensors, 175
SensorWrapper, 171
SEQ
protocol.h, 239
SET
protocol.h, 239
set_alert_latch
Alert Functions, 70
set_averaging_mode
Configuration Functions, 61
set_bus_conversion_time
Configuration Functions, 61
set_bus_measurement_disable
Configuration Functions, 60
set_bus_measurement_enable
Configuration Functions, 60
set_clock_sync_interval
DS3231, 114
set_crit_alert_limit
Alert Functions, 66
set_mode_continuous
Configuration Functions, 58
set_mode_power_down
Configuration Functions, 57
set_mode_triggered
Configuration Functions, 58
set_power_valid_limit
Alert Functions, 67
set_shunt_conversion_time
Configuration Functions, 62
set_shunt_measurement_disable
Configuration Functions, 59
set_shunt_measurement_enable
Configuration Functions, 59
set_telemetry_flush_threshold
Telemetry Manager, 77
set_telemetry_sample_interval
Telemetry Manager, 77
set_time
DS3231, 106
set_timezone_offset
DS3231, 113
set_unix_time
DS3231, 110
set_warn_alert_limit
Alert Functions, 66
setUp
test_comand_handlers.cpp, 368
test_error_codes.cpp, 375
test_frame_send.cpp, 384
shunt_conv_time
INA3221::conf_reg_t, 103
shunt_sum_alert
INA3221::masken_reg_t, 155
shunt_sum_en_ch1
INA3221::masken_reg_t, 156
shunt_sum_en_ch2
INA3221::masken_reg_t, 156
shunt_sum_en_ch3
INA3221::masken_reg_t, 156
SHUNT_VOLTAGE_LSB_UV
INA3221.h, 294
SHUTDOWN
Event Manager, 49
SILENT
utils.h, 356
SOLAR_ACTIVE
Event Manager, 49
SOLAR_CURRENT_THRESHOLD
PowerManager, 166
SOLAR_INACTIVE
Event Manager, 49
speed
TelemetryRecord, 178
spi_output_buffer
hardware_mocks.cpp, 387
hardware_mocks.h, 390
SPI_PORT
pin_config.h, 282
split_and_send_message
communication.h, 233
send.cpp, 252
splitString
gps_collector.cpp, 269
Storage Commands, 37
handle_list_files, 38
handle_mount, 38
storage.cpp
fs_init, 329
sd_card_mounted, 330
storage.h
fs_close_file, 333
fs_file_exists, 333
fs_init, 331
fs_open_file, 332
fs_read_file, 332
fs_write_file, 332
sd_card_mounted, 333
storage_commands.cpp
LIST_FILES_COMMAND, 219
MOUNT_COMMAND, 219
STORAGE_GROUP, 219
STORAGE_GROUP
storage_commands.cpp, 219
string_to_operation_type
protocol.h, 243
utils_converters.cpp, 257

SUNDAY
DS3231.h, 191
SX1278_CS
pin_config.h, 282
SX1278_MISO
pin_config.h, 282
SX1278_MOSI
pin_config.h, 282
SX1278_SCK
pin_config.h, 282
sync_clock_with_gps
DS3231, 117
sync_interval_minutes_
DS3231, 123
SYSTEM
Event Manager, 48
system_voltage
TelemetryRecord, 176
systemClock
Clock Management Commands, 15
Event Manager, 52
main.cpp, 363
telemetry_manager.cpp, 336
SystemEvent
Event Manager, 48

t_fine
BME280, 91
tearDown
test_comand_handlers.cpp, 368
test_error_codes.cpp, 375
test_frame_send.cpp, 384
Telemetry Buffer Commands, 39
handle_get_last_sensor_record, 40
handle_get_last_telemetry_record, 40
Telemetry Manager, 71
collect_telemetry, 73
flush_sensor_data, 74
flush_telemetry, 73
get_last_sensor_record_csv, 78
get_last_telemetry_record_csv, 77
get_telemetry_flush_threshold, 77
get_telemetry_sample_interval, 76
is_telemetry_collection_time, 75
is_telemetry_flush_time, 76
sensor_data_buffer, 80
sensor_data_buffer_count, 80
sensor_data_buffer_write_index, 80
set_telemetry_flush_threshold, 77
set_telemetry_sample_interval, 77
telemetry_buffer, 79
telemetry_buffer_count, 79
TELEMETRY_BUFFER_SIZE, 79, 80
telemetry_buffer_write_index, 79
telemetry_init, 72
telemetry_mutex, 80
telemetry_buffer
Telemetry Manager, 79
telemetry_buffer_count
Telemetry Manager, 79
telemetry_commands.cpp, 222
TELEMETRY_BUFFER_SIZE
Telemetry Manager, 79, 80
telemetry_buffer_write_index
Telemetry Manager, 79
telemetry_commands.cpp, 223
telemetry_commands.cpp
GET_LAST_SENSOR_RECORD_COMMAND, 222
GET_LAST_TELEMETRY_RECORD_COMMAND, 222
telemetry_buffer_count, 222
telemetry_buffer_write_index, 223
TELEMETRY_GROUP, 222
telemetry_mutex, 222
TELEMETRY_CSV_PATH
telemetry_manager.cpp, 336
TELEMETRY_GROUP
telemetry_commands.cpp, 222
telemetry_init
Telemetry Manager, 72
telemetry_manager.cpp
DEFAULT_FLUSH_THRESHOLD, 336
DEFAULT_SAMPLE_INTERVAL_MS, 336
flush_threshold, 337
nmea_data, 337
powerManager, 336
sample_interval_ms, 337
SENSOR_DATA_CSV_PATH, 336
systemClock, 336
TELEMETRY_CSV_PATH, 336
telemetry_mutex
Telemetry Manager, 80
telemetry_commands.cpp, 222
TelemetryRecord, 175
altitude, 179
battery_voltage, 176
build_version, 176
charge_current_solar, 177
charge_current_usb, 176
course, 178
date, 178
discharge_current, 177
fix_quality, 178
lat_dir, 177
latitude, 177
lon_dir, 178
longitude, 177
satellites, 178
speed, 178
system_voltage, 176
time, 177
timestamp, 176
to_csv, 176
TEMPERATURE
ISensor.h, 324
temperature

SensorDataRecord, 169
test/comms/test_comand_handlers.cpp, 366, 371
test/comms/test_converters.cpp, 372, 374
test/comms/test_error_codes.cpp, 375, 376
test/comms/test_frame_build.cpp, 377, 379
test/comms/test_frame_coding.cpp, 379, 381
test/comms/test_frame_common.h, 382, 383
test/comms/test_frame_send.cpp, 384, 385
test/mocks/hardware_mocks.cpp, 385, 387
test/mocks/hardware_mocks.h, 388, 390
test/test_runner.cpp, 390, 400
test_comand_handlers.cpp
last_frame_sent, 371
lora_send_called, 370
send_frame_lora, 368
send_frame_uart, 367
setUp, 368
tearDown, 368
test_command_handler_get_operation, 369
test_command_handler_invalid_operation, 370
test_command_handler_set_operation, 369
uart_send_called, 370
test_command_handler_get_operation
test_comand_handlers.cpp, 369
test_runner.cpp, 397
test_command_handler_invalid_operation
test_comand_handlers.cpp, 370
test_runner.cpp, 398
test_command_handler_set_operation
test_comand_handlers.cpp, 369
test_runner.cpp, 398
test_converters.cpp
test_exception_type_conversion, 373
test_hex_string_conversion, 374
test_operation_type_conversion, 372
test_value_unit_type_conversion, 372
test_error_code_conversion
test_error_codes.cpp, 376
test_runner.cpp, 399
test_error_codes.cpp
setUp, 375
tearDown, 375
test_error_code_conversion, 376
test_exception_type_conversion
test_converters.cpp, 373
test_runner.cpp, 396
test_frame_build.cpp
test_frame_build_error, 377
test_frame_build_info, 378
test_frame_build_success, 377
test_frame_build_error
test_frame_build.cpp, 377
test_runner.cpp, 393
test_frame_build_info
test_frame_build.cpp, 378
test_runner.cpp, 394
test_frame_build_success
test_frame_build.cpp, 377
test_runner.cpp, 393
test_frame_coding.cpp
test_frame_decode_basic, 380
test_frame_decode_invalid_header, 381
test_frame_encode_basic, 380
test_frame_common.h
create_test_frame, 383
test_frame_decode_basic
test_frame_coding.cpp, 380
test_runner.cpp, 391
test_frame_decode_invalid_header
test_frame_coding.cpp, 381
test_runner.cpp, 392
test_frame_encode_basic
test_frame_coding.cpp, 380
test_runner.cpp, 391
test_frame_send.cpp
setUp, 384
tearDown, 384
test_send_frame_uart, 384
test_hex_string_conversion
test_converters.cpp, 374
test_runner.cpp, 396
test_operation_type_conversion
test_converters.cpp, 372
test_runner.cpp, 394
test_runner.cpp
main, 400
test_command_handler_get_operation, 397
test_command_handler_invalid_operation, 398
test_command_handler_set_operation, 398
test_error_code_conversion, 399
test_exception_type_conversion, 396
test_frame_build_error, 393
test_frame_build_info, 394
test_frame_build_success, 393
test_frame_decode_basic, 391
test_frame_decode_invalid_header, 392
test_frame_encode_basic, 391
test_hex_string_conversion, 396
test_operation_type_conversion, 394
test_value_unit_type_conversion, 395
test_send_frame_uart
test_frame_send.cpp, 384
test_value_unit_type_conversion
test_converters.cpp, 372
test_runner.cpp, 395
TEXT
protocol.h, 240
THURSDAY
DS3231.h, 191
TIME
clock_commands.cpp, 194
time
TelemetryRecord, 177
timestamp
event_manager.h, 265
EventLog, 128

SensorDataRecord, 169
TelemetryRecord, 176
TIMEZONE_OFFSET
clock_commands.cpp, 194
timezone_offset_minutes_
DS3231, 123
timing_ctrl_alert
INA3221::masken_reg_t, 155
to_csv
SensorDataRecord, 168
TelemetryRecord, 176
to_string
event_manager.h, 265
EventLog, 128
TUESDAY
DS3231.h, 191

UART
protocol.h, 240
UART_ERROR
Event Manager, 49
uart_mutex
utils.cpp, 351
uart_output_buffer
hardware_mocks.cpp, 387
hardware_mocks.h, 389
uart_print
utils.cpp, 348
utils.h, 356
uart_send_called
test_comand_handlers.cpp, 370
UNCONFIGURED_POWER_DOWN
BH1750, 82
UNDEFINED
protocol.h, 240
unit
Frame, 140
UNKNOWN_ERROR
protocol.h, 239
update_gga_tokens
NMEAData, 160
update_last_sync_time
DS3231, 115
update_rmc_tokens
NMEAData, 160
USB_CONNECTED
Event Manager, 49
USB_CURRENT_THRESHOLD
PowerManager, 166
USB_DISCONNECTED
Event Manager, 49
utils.cpp
base64_chars, 351
base64_encode, 350
calculate_checksum, 350
g_uart_verbosity, 351
get_level_color, 346
get_level_prefix, 347
uart_mutex, 351
uart_print, 348
utils.h
ANSI_BLUE, 355
ANSI_CYAN, 355
ANSI_GREEN, 355
ANSI_RED, 355
ANSI_RESET, 355
ANSI_YELLOW, 355
base64_encode, 358
calculate_checksum, 358
DEBUG, 356
ERROR, 356
EVENT, 356
g_uart_verbosity, 359
INFO, 356
SILENT, 356
uart_print, 356
VerbosityLevel, 355
WARNING, 356
utils_converters.cpp
error_code_to_string, 257
exception_type_to_string, 254
hex_string_to_bytes, 259
operation_type_to_string, 256
string_to_operation_type, 257
value_unit_type_to_string, 255

VAL
protocol.h, 239
value
Frame, 140
value_unit_type_to_string
protocol.h, 244
utils_converters.cpp, 255
ValueUnit
protocol.h, 239
VerbosityLevel
utils.h, 355
VOLT
protocol.h, 240
VOLTAGE_BATTERY
power_commands.cpp, 210
VOLTAGE_LOW_THRESHOLD
PowerManager, 166
VOLTAGE_MAIN
power_commands.cpp, 210
VOLTAGE_OVERCHARGE_THRESHOLD
PowerManager, 167

warn_alert_ch1
INA3221::masken_reg_t, 155
warn_alert_ch2
INA3221::masken_reg_t, 155
warn_alert_ch3
INA3221::masken_reg_t, 155
warn_alert_latch_en
INA3221::masken_reg_t, 156
WARNING
utils.h, 356

WATCHDOG_RESET
Event Manager, 49
WEDNESDAY
DS3231.h, 191
write8
BH1750, 83
WRITE_ONLY
protocol.h, 239
write_register
HMC5883L, 142
writeIndex
EventManager, 133

year
ds3231_data_t, 125