KubiSat Firmware

Generated by Doxygen 1.13.2

1 Clock Commands 1

2 Topic Index 3

2.1 Topics . . . 3

3 Hierarchical Index 5

3.1 Class Hierarchy . . . 5

4 Class Index 7

4.1 Class List . . . 7

5 File Index 9

5.1 File List . . . 9

6 Topic Documentation 11

6.1 Clock Management Commands . . . 11

6.1.1 Detailed Description . . . 11

6.1.2 Function Documentation . . . 11

6.1.2.1 handle_time() . . . 11

6.1.2.2 handle_timezone_offset() . . . 12

6.1.2.3 handle_clock_sync_interval() . . . 13

6.1.2.4 handle_get_last_sync_time() . . . 14

6.1.3 Variable Documentation . . . 14

6.1.3.1 systemClock . . . 14

6.2 Command System . . . 14

6.2.1 Detailed Description . . . 15

6.2.2 Typedef Documentation . . . 15

6.2.2.1 CommandHandler . . . 15

6.2.2.2 CommandMap . . . 15

6.2.3 Function Documentation . . . 15

6.2.3.1 execute_command() . . . 15

6.2.4 Variable Documentation . . . 16

6.2.4.1 commandHandlers . . . 16

6.3 Diagnostic Commands . . . 17

6.3.1 Detailed Description . . . 17

6.3.2 Function Documentation . . . 17

6.3.2.1 handle_get_commands_list() . . . 17

6.3.2.2 handle_get_build_version() . . . 18

6.3.2.3 handle_verbosity() . . . 19

6.3.2.4 handle_enter_bootloader_mode() . . . 20

6.4 Event Commands . . . 21

6.4.1 Detailed Description . . . 21

6.4.2 Function Documentation . . . 21

6.4.2.1 handle_get_last_events() . . . 21

6.4.2.2 handle_get_event_count() . . . 22
6.5 GPS Commands . . . 23
6.5.1 Detailed Description . . . 23
6.5.2 Function Documentation . . . 23
6.5.2.1 handle_gps_power_status() . . . 23
6.5.2.2 handle_enable_gps_uart_passthrough() . . . 24
6.5.2.3 handle_get_rmc_data() . . . 25
6.5.2.4 handle_get_gga_data() . . . 26
6.6 Power Commands . . . 27
6.6.1 Detailed Description . . . 27
6.6.2 Function Documentation . . . 27
6.6.2.1 handle_get_power_manager_ids() . . . 27
6.6.2.2 handle_get_voltage_battery() . . . 28
6.6.2.3 handle_get_voltage_5v() . . . 29
6.6.2.4 handle_get_current_charge_usb() . . . 29
6.6.2.5 handle_get_current_charge_solar() . . . 30
6.6.2.6 handle_get_current_charge_total() . . . 31
6.6.2.7 handle_get_current_draw() . . . 32
6.7 Storage Commands . . . 33
6.7.1 Detailed Description . . . 33
6.7.2 Function Documentation . . . 33
6.7.2.1 handle_file_download() . . . 33
6.7.2.2 handle_list_files() . . . 34
6.7.2.3 handle_mount() . . . 35
6.8 INA3221 Power Monitor . . . 36
6.8.1 Detailed Description . . . 36
6.8.2 Configuration Functions . . . 36
6.8.2.1 Detailed Description . . . 37
6.8.2.2 Function Documentation . . . 37
6.8.3 Measurement Functions . . . 46
6.8.3.1 Detailed Description . . . 46
6.8.3.2 Function Documentation . . . 46
6.8.4 Alert Functions . . . 48
6.8.4.1 Detailed Description . . . 49
6.8.4.2 Function Documentation . . . 49

7 Class Documentation **55**
7.1 BH1750 Class Reference . . . 55
7.1.1 Detailed Description . . . 55
7.1.2 Member Enumeration Documentation . . . 55
7.1.2.1 Mode . . . 55
7.1.3 Constructor & Destructor Documentation . . . 56

7.1.3.1 BH1750() . . . 56
7.1.4 Member Function Documentation . . . 56
7.1.4.1 begin() . . . 56
7.1.4.2 configure() . . . 57
7.1.4.3 get_light_level() . . . 57
7.1.4.4 write8() . . . 57
7.1.5 Member Data Documentation . . . 58
7.1.5.1 _i2c_addr . . . 58
7.2 BH1750Wrapper Class Reference . . . 58
7.2.1 Detailed Description . . . 59
7.2.2 Constructor & Destructor Documentation . . . 59
7.2.2.1 BH1750Wrapper() . . . 59
7.2.3 Member Function Documentation . . . 59
7.2.3.1 get_i2c_addr() . . . 59
7.2.3.2 init() . . . 59
7.2.3.3 read_data() . . . 60
7.2.3.4 is_initialized() . . . 60
7.2.3.5 get_type() . . . 60
7.2.3.6 configure() . . . 60
7.2.4 Member Data Documentation . . . 60
7.2.4.1 sensor_ . . . 60
7.2.4.2 initialized_ . . . 60
7.3 BME280 Class Reference . . . 61
7.3.1 Detailed Description . . . 62
7.3.2 Constructor & Destructor Documentation . . . 62
7.3.2.1 BME280() . . . 62
7.3.3 Member Function Documentation . . . 63
7.3.3.1 init() . . . 63
7.3.3.2 reset() . . . 63
7.3.3.3 read_raw_all() . . . 63
7.3.3.4 convert_temperature() . . . 63
7.3.3.5 convert_pressure() . . . 63
7.3.3.6 convert_humidity() . . . 64
7.3.3.7 configure_sensor() . . . 64
7.3.3.8 get_calibration_parameters() . . . 64
7.3.4 Member Data Documentation . . . 64
7.3.4.1 ADDR_SDO_LOW . . . 64
7.3.4.2 ADDR_SDO_HIGH . . . 65
7.3.4.3 i2c_port . . . 65
7.3.4.4 device_addr . . . 65
7.3.4.5 calib_params . . . 65
7.3.4.6 initialized_ . . . 65

7.3.4.7 t_fine . . . 65
7.3.4.8 REG_CONFIG . . . 65
7.3.4.9 REG_CTRL_MEAS . . . 65
7.3.4.10 REG_CTRL_HUM . . . 66
7.3.4.11 REG_RESET . . . 66
7.3.4.12 REG_PRESSURE_MSB . . . 66
7.3.4.13 REG_TEMPERATURE_MSB . . . 66
7.3.4.14 REG_HUMIDITY_MSB . . . 66
7.3.4.15 REG_DIG_T1_LSB . . . 66
7.3.4.16 REG_DIG_T1_MSB . . . 66
7.3.4.17 REG_DIG_T2_LSB . . . 66
7.3.4.18 REG_DIG_T2_MSB . . . 67
7.3.4.19 REG_DIG_T3_LSB . . . 67
7.3.4.20 REG_DIG_T3_MSB . . . 67
7.3.4.21 REG_DIG_P1_LSB . . . 67
7.3.4.22 REG_DIG_P1_MSB . . . 67
7.3.4.23 REG_DIG_P2_LSB . . . 67
7.3.4.24 REG_DIG_P2_MSB . . . 67
7.3.4.25 REG_DIG_P3_LSB . . . 67
7.3.4.26 REG_DIG_P3_MSB . . . 68
7.3.4.27 REG_DIG_P4_LSB . . . 68
7.3.4.28 REG_DIG_P4_MSB . . . 68
7.3.4.29 REG_DIG_P5_LSB . . . 68
7.3.4.30 REG_DIG_P5_MSB . . . 68
7.3.4.31 REG_DIG_P6_LSB . . . 68
7.3.4.32 REG_DIG_P6_MSB . . . 68
7.3.4.33 REG_DIG_P7_LSB . . . 68
7.3.4.34 REG_DIG_P7_MSB . . . 69
7.3.4.35 REG_DIG_P8_LSB . . . 69
7.3.4.36 REG_DIG_P8_MSB . . . 69
7.3.4.37 REG_DIG_P9_LSB . . . 69
7.3.4.38 REG_DIG_P9_MSB . . . 69
7.3.4.39 REG_DIG_H1 . . . 69
7.3.4.40 REG_DIG_H2 . . . 69
7.3.4.41 REG_DIG_H3 . . . 69
7.3.4.42 REG_DIG_H4 . . . 70
7.3.4.43 REG_DIG_H5 . . . 70
7.3.4.44 REG_DIG_H6 . . . 70
7.3.4.45 NUM_CALIB_PARAMS . . . 70
7.4 BME280CalibParam Struct Reference . . . 70
7.4.1 Detailed Description . . . 71
7.4.2 Member Data Documentation . . . 71

7.4.2.1 dig_t1 . . . 71
7.4.2.2 dig_t2 . . . 71
7.4.2.3 dig_t3 . . . 71
7.4.2.4 dig_p1 . . . 71
7.4.2.5 dig_p2 . . . 71
7.4.2.6 dig_p3 . . . 71
7.4.2.7 dig_p4 . . . 72
7.4.2.8 dig_p5 . . . 72
7.4.2.9 dig_p6 . . . 72
7.4.2.10 dig_p7 . . . 72
7.4.2.11 dig_p8 . . . 72
7.4.2.12 dig_p9 . . . 72
7.4.2.13 dig_h1 . . . 72
7.4.2.14 dig_h2 . . . 72
7.4.2.15 dig_h3 . . . 73
7.4.2.16 dig_h4 . . . 73
7.4.2.17 dig_h5 . . . 73
7.4.2.18 dig_h6 . . . 73
7.5 BME280Wrapper Class Reference . . . 73
7.5.1 Detailed Description . . . 74
7.5.2 Constructor & Destructor Documentation . . . 74
7.5.2.1 BME280Wrapper() . . . 74
7.5.3 Member Function Documentation . . . 75
7.5.3.1 init() . . . 75
7.5.3.2 read_data() . . . 75
7.5.3.3 is_initialized() . . . 75
7.5.3.4 get_type() . . . 75
7.5.3.5 configure() . . . 75
7.5.4 Member Data Documentation . . . 75
7.5.4.1 sensor_ . . . 75
7.5.4.2 initialized_ . . . 76
7.6 INA3221::conf_reg_t Struct Reference . . . 76
7.6.1 Detailed Description . . . 76
7.6.2 Member Data Documentation . . . 76
7.6.2.1 mode_shunt_en . . . 76
7.6.2.2 mode_bus_en . . . 76
7.6.2.3 mode_continious_en . . . 77
7.6.2.4 shunt_conv_time . . . 77
7.6.2.5 bus_conv_time . . . 77
7.6.2.6 avg_mode . . . 77
7.6.2.7 ch3_en . . . 77
7.6.2.8 ch2_en . . . 77

7.6.2.9 ch1_en . . . 77
7.6.2.10 reset . . . 78
7.7 DS3231 Class Reference . . . 78
7.7.1 Detailed Description . . . 78
7.7.2 Constructor & Destructor Documentation . . . 79
7.7.2.1 DS3231() . . . 79
7.7.3 Member Function Documentation . . . 79
7.7.3.1 set_time() . . . 79
7.7.3.2 get_time() . . . 80
7.7.3.3 read_temperature() . . . 80
7.7.3.4 set_unix_time() . . . 81
7.7.3.5 get_unix_time() . . . 81
7.7.3.6 clock_enable() . . . 81
7.7.3.7 i2c_read_reg() . . . 82
7.7.3.8 i2c_write_reg() . . . 83
7.7.3.9 bin_to_bcd() . . . 83
7.7.3.10 bcd_to_bin() . . . 84
7.7.4 Member Data Documentation . . . 85
7.7.4.1 i2c . . . 85
7.7.4.2 ds3231_addr . . . 85
7.7.4.3 clock_mutex_ . . . 85
7.8 ds3231_data_t Struct Reference . . . 85
7.8.1 Detailed Description . . . 85
7.8.2 Member Data Documentation . . . 86
7.8.2.1 seconds . . . 86
7.8.2.2 minutes . . . 86
7.8.2.3 hours . . . 86
7.8.2.4 day . . . 86
7.8.2.5 date . . . 86
7.8.2.6 month . . . 86
7.8.2.7 year . . . 86
7.8.2.8 century . . . 87
7.9 EventEmitter Class Reference . . . 87
7.9.1 Detailed Description . . . 87
7.9.2 Member Function Documentation . . . 87
7.9.2.1 emit() . . . 87
7.10 EventLog Class Reference . . . 88
7.10.1 Detailed Description . . . 89
7.10.2 Member Function Documentation . . . 89
7.10.2.1 to_string() . . . 89
7.10.3 Member Data Documentation . . . 89
7.10.3.1 id . . . 89

7.10.3.2 timestamp . . . 89
7.10.3.3 group . . . 90
7.10.3.4 event . . . 90
7.11 EventManager Class Reference . . . 90
7.11.1 Detailed Description . . . 91
7.11.2 Constructor & Destructor Documentation . . . 91
7.11.2.1 EventManager() . . . 91
7.11.2.2 ~EventManager() . . . 92
7.11.3 Member Function Documentation . . . 92
7.11.3.1 init() . . . 92
7.11.3.2 log_event() . . . 92
7.11.3.3 get_event() . . . 93
7.11.3.4 get_event_count() . . . 94
7.11.3.5 save_to_storage() . . . 94
7.11.3.6 load_from_storage() . . . 94
7.11.4 Member Data Documentation . . . 95
7.11.4.1 events . . . 95
7.11.4.2 eventCount . . . 95
7.11.4.3 writeIndex . . . 95
7.11.4.4 eventMutex . . . 95
7.11.4.5 nextEventId . . . 95
7.11.4.6 needsPersistence . . . 95
7.12 EventManagerImpl Class Reference . . . 96
7.12.1 Detailed Description . . . 97
7.12.2 Constructor & Destructor Documentation . . . 97
7.12.2.1 EventManagerImpl() . . . 97
7.12.3 Member Function Documentation . . . 98
7.12.3.1 save_to_storage() . . . 98
7.12.3.2 load_from_storage() . . . 98
7.13 FileHandle Struct Reference . . . 99
7.13.1 Detailed Description . . . 99
7.13.2 Member Data Documentation . . . 99
7.13.2.1 fd . . . 99
7.13.2.2 is_open . . . 99
7.14 Frame Struct Reference . . . 99
7.14.1 Detailed Description . . . 100
7.14.2 Member Data Documentation . . . 100
7.14.2.1 header . . . 100
7.14.2.2 direction . . . 100
7.14.2.3 operationType . . . 100
7.14.2.4 group . . . 100
7.14.2.5 command . . . 100

7.14.2.6 value . . . 100
7.14.2.7 unit . . . 101
7.14.2.8 footer . . . 101
7.15 HMC5883L Class Reference . . . 101
7.15.1 Detailed Description . . . 101
7.15.2 Constructor & Destructor Documentation . . . 101
7.15.2.1 HMC5883L() . . . 101
7.15.3 Member Function Documentation . . . 102
7.15.3.1 init() . . . 102
7.15.3.2 read() . . . 102
7.15.3.3 write_register() . . . 102
7.15.3.4 read_register() . . . 103
7.15.4 Member Data Documentation . . . 103
7.15.4.1 i2c . . . 103
7.15.4.2 address . . . 103
7.16 HMC5883LWrapper Class Reference . . . 104
7.16.1 Detailed Description . . . 105
7.16.2 Constructor & Destructor Documentation . . . 105
7.16.2.1 HMC5883LWrapper() . . . 105
7.16.3 Member Function Documentation . . . 105
7.16.3.1 init() . . . 105
7.16.3.2 read_data() . . . 105
7.16.3.3 is_initialized() . . . 105
7.16.3.4 get_type() . . . 106
7.16.3.5 configure() . . . 106
7.16.4 Member Data Documentation . . . 106
7.16.4.1 sensor_ . . . 106
7.16.4.2 initialized_ . . . 106
7.17 INA3221 Class Reference . . . 106
7.17.1 Detailed Description . . . 108
7.17.2 Member Function Documentation . . . 108
7.17.2.1 _read() . . . 108
7.17.2.2 _write() . . . 110
7.17.2.3 get_current() . . . 111
7.17.3 Member Data Documentation . . . 112
7.17.3.1 _i2c . . . 112
7.17.3.2 _i2c_addr . . . 112
7.17.3.3 _shuntRes . . . 112
7.17.3.4 _filterRes . . . 112
7.17.3.5 _masken_reg . . . 112
7.18 ISensor Class Reference . . . 112
7.18.1 Detailed Description . . . 113

7.18.2 Constructor & Destructor Documentation . . . 113
7.18.2.1 ~ISensor() . . . 113
7.18.3 Member Function Documentation . . . 113
7.18.3.1 init() . . . 113
7.18.3.2 read_data() . . . 113
7.18.3.3 is_initialized() . . . 113
7.18.3.4 get_type() . . . 113
7.18.3.5 configure() . . . 114
7.19 LoRaClass Class Reference . . . 114
7.19.1 Detailed Description . . . 117
7.19.2 Constructor & Destructor Documentation . . . 117
7.19.2.1 LoRaClass() . . . 117
7.19.3 Member Function Documentation . . . 117
7.19.3.1 begin() . . . 117
7.19.3.2 end() . . . 118
7.19.3.3 beginPacket() . . . 118
7.19.3.4 endPacket() . . . 118
7.19.3.5 parse_packet() . . . 119
7.19.3.6 packetRssi() . . . 119
7.19.3.7 packetSnr() . . . 119
7.19.3.8 packetFrequencyError() . . . 120
7.19.3.9 rssi() . . . 120
7.19.3.10 write() [1/2] . . . 120
7.19.3.11 write() [2/2] . . . 121
7.19.3.12 available() . . . 121
7.19.3.13 read() . . . 122
7.19.3.14 peek() . . . 122
7.19.3.15 flush() . . . 123
7.19.3.16 onCadDone() . . . 123
7.19.3.17 onReceive() . . . 123
7.19.3.18 onTxDone() . . . 124
7.19.3.19 receive() . . . 124
7.19.3.20 channelActivityDetection() . . . 124
7.19.3.21 idle() . . . 125
7.19.3.22 sleep() . . . 125
7.19.3.23 setTxPower() . . . 126
7.19.3.24 setFrequency() . . . 126
7.19.3.25 setSpreadingFactor() . . . 127
7.19.3.26 setSignalBandwidth() . . . 127
7.19.3.27 setCodingRate4() . . . 128
7.19.3.28 setPreambleLength() . . . 128
7.19.3.29 setSyncWord() . . . 128

7.19.3.30 enableCrc() . . . 129
7.19.3.31 disableCrc() . . . 129
7.19.3.32 enableInvertIQ() . . . 130
7.19.3.33 disableInvertIQ() . . . 130
7.19.3.34 setOCP() . . . 130
7.19.3.35 setGain() . . . 131
7.19.3.36 crc() . . . 131
7.19.3.37 noCrc() . . . 131
7.19.3.38 random() . . . 132
7.19.3.39 set_pins() . . . 132
7.19.3.40 setSPI() . . . 132
7.19.3.41 setSPIFrequency() . . . 132
7.19.3.42 dumpRegisters() . . . 132
7.19.3.43 explicitHeaderMode() . . . 133
7.19.3.44 implicitHeaderMode() . . . 133
7.19.3.45 handleDio0Rise() . . . 134
7.19.3.46 isTransmitting() . . . 134
7.19.3.47 getSpreadingFactor() . . . 135
7.19.3.48 getSignalBandwidth() . . . 135
7.19.3.49 setLdoFlag() . . . 136
7.19.3.50 readRegister() . . . 136
7.19.3.51 writeRegister() . . . 138
7.19.3.52 singleTransfer() . . . 139
7.19.3.53 onDio0Rise() . . . 140
7.19.4 Member Data Documentation . . . 141
7.19.4.1 _spi . . . 141
7.19.4.2 _ss . . . 141
7.19.4.3 _reset . . . 141
7.19.4.4 _dio0 . . . 141
7.19.4.5 _frequency . . . 141
7.19.4.6 _packetIndex . . . 142
7.19.4.7 _implicitHeaderMode . . . 142
7.19.4.8 _onReceive . . . 142
7.19.4.9 _onCadDone . . . 142
7.19.4.10 _onTxDone . . . 142
7.20 INA3221::masken_reg_t Struct Reference . . . 142
7.20.1 Detailed Description . . . 143
7.20.2 Member Data Documentation . . . 143
7.20.2.1 conv_ready . . . 143
7.20.2.2 timing_ctrl_alert . . . 143
7.20.2.3 pwr_valid_alert . . . 143
7.20.2.4 warn_alert_ch3 . . . 143

7.20.2.5 warn_alert_ch2 . . . 144
7.20.2.6 warn_alert_ch1 . . . 144
7.20.2.7 shunt_sum_alert . . . 144
7.20.2.8 crit_alert_ch3 . . . 144
7.20.2.9 crit_alert_ch2 . . . 144
7.20.2.10 crit_alert_ch1 . . . 144
7.20.2.11 crit_alert_latch_en . . . 144
7.20.2.12 warn_alert_latch_en . . . 144
7.20.2.13 shunt_sum_en_ch3 . . . 145
7.20.2.14 shunt_sum_en_ch2 . . . 145
7.20.2.15 shunt_sum_en_ch1 . . . 145
7.20.2.16 reserved . . . 145
7.21 MPU6050Wrapper Class Reference . . . 145
7.21.1 Detailed Description . . . 146
7.21.2 Constructor & Destructor Documentation . . . 146
7.21.2.1 MPU6050Wrapper() . . . 146
7.21.3 Member Function Documentation . . . 147
7.21.3.1 init() . . . 147
7.21.3.2 read_data() . . . 147
7.21.3.3 is_initialized() . . . 147
7.21.3.4 get_type() . . . 147
7.21.3.5 configure() . . . 147
7.21.4 Member Data Documentation . . . 147
7.21.4.1 sensor_ . . . 147
7.21.4.2 initialized_ . . . 148
7.22 NMEAData Class Reference . . . 148
7.22.1 Detailed Description . . . 148
7.22.2 Constructor & Destructor Documentation . . . 148
7.22.2.1 NMEAData() . . . 148
7.22.3 Member Function Documentation . . . 149
7.22.3.1 update_rmc_tokens() . . . 149
7.22.3.2 update_gga_tokens() . . . 149
7.22.3.3 get_rmc_tokens() . . . 149
7.22.3.4 get_gga_tokens() . . . 149
7.22.4 Member Data Documentation . . . 149
7.22.4.1 rmc_tokens_ . . . 149
7.22.4.2 gga_tokens_ . . . 149
7.22.4.3 rmc_mutex_ . . . 149
7.22.4.4 gga_mutex_ . . . 150
7.23 PowerManager Class Reference . . . 150
7.23.1 Detailed Description . . . 151
7.23.2 Constructor & Destructor Documentation . . . 151

7.23.2.1 PowerManager() . . . 151
7.23.3 Member Function Documentation . . . 151
7.23.3.1 initialize() . . . 151
7.23.3.2 read_device_ids() . . . 151
7.23.3.3 get_current_charge_solar() . . . 152
7.23.3.4 get_current_charge_usb() . . . 152
7.23.3.5 get_current_charge_total() . . . 152
7.23.3.6 get_current_draw() . . . 152
7.23.3.7 get_voltage_battery() . . . 152
7.23.3.8 get_voltage_5v() . . . 153
7.23.3.9 configure() . . . 153
7.23.3.10 is_charging_solar() . . . 153
7.23.3.11 is_charging_usb() . . . 154
7.23.3.12 check_power_alerts() . . . 154
7.23.4 Member Data Documentation . . . 154
7.23.4.1 SOLAR_CURRENT_THRESHOLD . . . 154
7.23.4.2 USB_CURRENT_THRESHOLD . . . 155
7.23.4.3 VOLTAGE_LOW_THRESHOLD . . . 155
7.23.4.4 VOLTAGE_OVERCHARGE_THRESHOLD . . . 155
7.23.4.5 FALL_RATE_THRESHOLD . . . 155
7.23.4.6 FALLING_TREND_REQUIRED . . . 155
7.23.4.7 ina3221_ . . . 155
7.23.4.8 initialized_ . . . 155
7.23.4.9 powerman_mutex_ . . . 155
7.23.4.10 charging_solar_active_ . . . 156
7.23.4.11 charging_usb_active_ . . . 156
7.24 Print Class Reference . . . 156
7.24.1 Detailed Description . . . 157
7.24.2 Constructor & Destructor Documentation . . . 157
7.24.2.1 Print() . . . 157
7.24.3 Member Function Documentation . . . 158
7.24.3.1 printNumber() . . . 158
7.24.3.2 printULLNumber() . . . 158
7.24.3.3 printFloat() . . . 159
7.24.3.4 setWriteError() . . . 160
7.24.3.5 getWriteError() . . . 160
7.24.3.6 clearWriteError() . . . 160
7.24.3.7 write() [1/4] . . . 161
7.24.3.8 write() [2/4] . . . 161
7.24.3.9 write() [3/4] . . . 162
7.24.3.10 write() [4/4] . . . 162
7.24.3.11 availableForWrite() . . . 163

7.24.3.12 print() [1/11] . . . 163
7.24.3.13 print() [2/11] . . . 164
7.24.3.14 print() [3/11] . . . 165
7.24.3.15 print() [4/11] . . . 165
7.24.3.16 print() [5/11] . . . 166
7.24.3.17 print() [6/11] . . . 166
7.24.3.18 print() [7/11] . . . 167
7.24.3.19 print() [8/11] . . . 167
7.24.3.20 print() [9/11] . . . 168
7.24.3.21 print() [10/11] . . . 168
7.24.3.22 print() [11/11] . . . 169
7.24.3.23 println() [1/11] . . . 169
7.24.3.24 println() [2/11] . . . 169
7.24.3.25 println() [3/11] . . . 170
7.24.3.26 println() [4/11] . . . 170
7.24.3.27 println() [5/11] . . . 171
7.24.3.28 println() [6/11] . . . 171
7.24.3.29 println() [7/11] . . . 172
7.24.3.30 println() [8/11] . . . 172
7.24.3.31 println() [9/11] . . . 173
7.24.3.32 println() [10/11] . . . 173
7.24.3.33 println() [11/11] . . . 174
7.24.3.34 flush() . . . 175
7.24.4 Member Data Documentation . . . 175
7.24.4.1 write_error . . . 175
7.25 SensorWrapper Class Reference . . . 176
7.25.1 Detailed Description . . . 176
7.25.2 Constructor & Destructor Documentation . . . 177
7.25.2.1 SensorWrapper() . . . 177
7.25.3 Member Function Documentation . . . 177
7.25.3.1 get_instance() . . . 177
7.25.3.2 sensor_init() . . . 177
7.25.3.3 sensor_configure() . . . 178
7.25.3.4 sensor_read_data() . . . 178
7.25.4 Member Data Documentation . . . 179
7.25.4.1 sensors . . . 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 . . . 187
8.7.1 Macro Definition Documentation . . . 188
8.7.1.1 DS3231_DEVICE_ADRESS . . . 188
8.7.1.2 DS3231_SECONDS_REG . . . 188
8.7.1.3 DS3231_MINUTES_REG . . . 188
8.7.1.4 DS3231_HOURS_REG . . . 188
8.7.1.5 DS3231_DAY_REG . . . 188
8.7.1.6 DS3231_DATE_REG . . . 188
8.7.1.7 DS3231_MONTH_REG . . . 189
8.7.1.8 DS3231_YEAR_REG . . . 189
8.7.1.9 DS3231_CONTROL_REG . . . 189
8.7.1.10 DS3231_CONTROL_STATUS_REG . . . 189
8.7.1.11 DS3231_TEMPERATURE_MSB_REG . . . 189
8.7.1.12 DS3231_TEMPERATURE_LSB_REG . . . 189
8.7.2 Enumeration Type Documentation . . . 189
8.7.2.1 days_of_week . . . 189
8.8 DS3231.h . . . 190
8.9 lib/comms/commands/clock_commands.cpp File Reference . . . 190
8.9.1 Macro Definition Documentation . . . 191
8.9.1.1 CLOCK_GROUP . . . 191
8.9.1.2 TIME . . . 191
8.9.1.3 TIMEZONE_OFFSET . . . 192
8.9.1.4 CLOCK_SYNC_INTERVAL . . . 192
8.9.1.5 LAST_SYNC_TIME . . . 192
8.10 clock_commands.cpp . . . 192
8.11 lib/comms/commands/commands.cpp File Reference . . . 194
8.12 commands.cpp . . . 194
8.13 lib/comms/commands/commands.h File Reference . . . 195
8.14 commands.h . . . 197
8.15 lib/comms/commands/diagnostic_commands.cpp File Reference . . . 198
8.16 diagnostic_commands.cpp . . . 198
8.17 lib/comms/commands/event_commands.cpp File Reference . . . 199
8.18 event_commands.cpp . . . 200
8.19 lib/comms/commands/gps_commands.cpp File Reference . . . 201
8.20 gps_commands.cpp . . . 201
8.21 lib/comms/commands/power_commands.cpp File Reference . . . 203
8.22 power_commands.cpp . . . 204
8.23 lib/comms/commands/storage_commands.cpp File Reference . . . 205

8.23.1 Macro Definition Documentation . . . 206
8.23.1.1 MAX_BLOCK_SIZE . . . 206
8.23.1.2 STORAGE_GROUP . . . 206
8.23.1.3 START_COMMAND . . . 206
8.23.1.4 DATA_COMMAND . . . 206
8.23.1.5 END_COMMAND . . . 207
8.23.1.6 LIST_FILES_COMMAND . . . 207
8.23.1.7 MOUNT_COMMAND . . . 207
8.24 storage_commands.cpp . . . 207
8.25 lib/comms/commands/storage_commands_utils.cpp File Reference . . . 209
8.25.1 Function Documentation . . . 209
8.25.1.1 calculate_checksum() . . . 209
8.25.1.2 send_data_block() . . . 210
8.25.1.3 receive_ack() . . . 210
8.26 storage_commands_utils.cpp . . . 211
8.27 lib/comms/commands/storage_commands_utils.h File Reference . . . 211
8.27.1 Function Documentation . . . 212
8.27.1.1 calculate_checksum() . . . 212
8.27.1.2 send_data_block() . . . 212
8.27.1.3 receive_ack() . . . 212
8.28 storage_commands_utils.h . . . 213
8.29 lib/comms/communication.cpp File Reference . . . 213
8.29.1 Function Documentation . . . 213
8.29.1.1 initialize_radio() . . . 213
8.29.2 Variable Documentation . . . 214
8.29.2.1 outgoing . . . 214
8.29.2.2 msgCount . . . 214
8.29.2.3 lastSendTime . . . 215
8.29.2.4 lastReceiveTime . . . 215
8.29.2.5 lastPrintTime . . . 215
8.29.2.6 interval . . . 215
8.30 communication.cpp . . . 215
8.31 lib/comms/communication.h File Reference . . . 216
8.31.1 Function Documentation . . . 217
8.31.1.1 initialize_radio() . . . 217
8.31.1.2 on_receive() . . . 217
8.31.1.3 handle_uart_input() . . . 218
8.31.1.4 send_message() . . . 219
8.31.1.5 send_frame() . . . 219
8.31.1.6 send_frame_uart() . . . 220
8.31.1.7 send_frame_lora() . . . 220
8.31.1.8 split_and_send_message() . . . 221

8.31.1.9 frame_process() . . . 221
8.31.1.10 frame_encode() . . . 222
8.31.1.11 frame_decode() . . . 223
8.31.1.12 frame_build() . . . 224
8.31.1.13 determine_unit() . . . 226
8.32 communication.h . . . 227
8.33 lib/comms/frame.cpp File Reference . . . 227
8.33.1 Detailed Description . . . 228
8.33.2 Typedef Documentation . . . 228
8.33.2.1 CommandHandler . . . 228
8.33.3 Function Documentation . . . 228
8.33.3.1 frame_encode() . . . 228
8.33.3.2 frame_decode() . . . 229
8.33.3.3 frame_process() . . . 230
8.33.3.4 frame_build() . . . 231
8.33.4 Variable Documentation . . . 232
8.33.4.1 eventRegister . . . 232
8.34 frame.cpp . . . 233
8.35 lib/comms/LoRa/LoRa-RP2040.cpp File Reference . . . 234
8.35.1 Macro Definition Documentation . . . 236
8.35.1.1 REG_FIFO . . . 236
8.35.1.2 REG_OP_MODE . . . 236
8.35.1.3 REG_FRF_MSB . . . 236
8.35.1.4 REG_FRF_MID . . . 236
8.35.1.5 REG_FRF_LSB . . . 236
8.35.1.6 REG_PA_CONFIG . . . 236
8.35.1.7 IRQ_CAD_DETECTED_MASK . . . 236
8.35.1.8 REG_OCP . . . 237
8.35.1.9 REG_LNA . . . 237
8.35.1.10 IRQ_CAD_DONE_MASK . . . 237
8.35.1.11 REG_FIFO_ADDR_PTR . . . 237
8.35.1.12 REG_FIFO_TX_BASE_ADDR . . . 237
8.35.1.13 REG_FIFO_RX_BASE_ADDR . . . 237
8.35.1.14 REG_FIFO_RX_CURRENT_ADDR . . . 237
8.35.1.15 REG_IRQ_FLAGS . . . 237
8.35.1.16 REG_RX_NB_BYTES . . . 238
8.35.1.17 REG_PKT_SNR_VALUE . . . 238
8.35.1.18 REG_PKT_RSSI_VALUE . . . 238
8.35.1.19 REG_RSSI_VALUE . . . 238
8.35.1.20 REG_MODEM_CONFIG_1 . . . 238
8.35.1.21 REG_MODEM_CONFIG_2 . . . 238
8.35.1.22 REG_PREAMBLE_MSB . . . 238

8.35.1.23 REG_PREAMBLE_LSB . . . 238
8.35.1.24 REG_PAYLOAD_LENGTH . . . 239
8.35.1.25 REG_MODEM_CONFIG_3 . . . 239
8.35.1.26 REG_FREQ_ERROR_MSB . . . 239
8.35.1.27 REG_FREQ_ERROR_MID . . . 239
8.35.1.28 REG_FREQ_ERROR_LSB . . . 239
8.35.1.29 REG_RSSI_WIDEBAND . . . 239
8.35.1.30 REG_DETECTION_OPTIMIZE . . . 239
8.35.1.31 REG_INVERTIQ . . . 239
8.35.1.32 REG_DETECTION_THRESHOLD . . . 240
8.35.1.33 REG_SYNC_WORD . . . 240
8.35.1.34 REG_INVERTIQ2 . . . 240
8.35.1.35 REG_DIO_MAPPING_1 . . . 240
8.35.1.36 REG_VERSION . . . 240
8.35.1.37 REG_PA_DAC . . . 240
8.35.1.38 MODE_CAD . . . 240
8.35.1.39 MODE_LONG_RANGE_MODE . . . 240
8.35.1.40 MODE_SLEEP . . . 241
8.35.1.41 MODE_STDBY . . . 241
8.35.1.42 MODE_TX . . . 241
8.35.1.43 MODE_RX_CONTINUOUS . . . 241
8.35.1.44 MODE_RX_SINGLE . . . 241
8.35.1.45 PA_BOOST . . . 241
8.35.1.46 IRQ_TX_DONE_MASK . . . 241
8.35.1.47 IRQ_PAYLOAD_CRC_ERROR_MASK . . . 241
8.35.1.48 IRQ_RX_DONE_MASK . . . 242
8.35.1.49 RF_MID_BAND_THRESHOLD . . . 242
8.35.1.50 RSSI_OFFSET_HF_PORT . . . 242
8.35.1.51 RSSI_OFFSET_LF_PORT . . . 242
8.35.1.52 MAX_PKT_LENGTH . . . 242
8.35.1.53 ISR_PREFIX . . . 242
8.35.2 Variable Documentation . . . 242
8.35.2.1 LoRa . . . 242
8.36 LoRa-RP2040.cpp . . . 243
8.37 lib/comms/LoRa/LoRa-RP2040.h File Reference . . . 251
8.37.1 Function Documentation . . . 252
8.37.1.1 __empty() . . . 252
8.37.2 Variable Documentation . . . 252
8.37.2.1 LoRa . . . 252
8.38 LoRa-RP2040.h . . . 253
8.39 lib/comms/LoRa/Print.cpp File Reference . . . 254
8.40 Print.cpp . . . 254

8.41 lib/comms/LoRa/Print.h File Reference . . . 259
8.41.1 Macro Definition Documentation . . . 259
8.41.1.1 DEC . . . 259
8.41.1.2 HEX . . . 260
8.41.1.3 OCT . . . 260
8.41.1.4 BIN . . . 260
8.42 Print.h . . . 260
8.43 lib/comms/protocol.h File Reference . . . 261
8.43.1 Enumeration Type Documentation . . . 262
8.43.1.1 ExecutionResult . . . 262
8.43.1.2 OperationType . . . 263
8.43.1.3 CommandAccessLevel . . . 263
8.43.1.4 ValueUnit . . . 263
8.43.1.5 ExceptionType . . . 263
8.43.1.6 Interface . . . 264
8.43.2 Function Documentation . . . 264
8.43.2.1 exception_type_to_string() . . . 264
8.43.2.2 operation_type_to_string() . . . 265
8.43.2.3 string_to_operation_type() . . . 266
8.43.2.4 hex_string_to_bytes() . . . 267
8.43.2.5 value_unit_type_to_string() . . . 267
8.43.3 Variable Documentation . . . 269
8.43.3.1 FRAME_BEGIN . . . 269
8.43.3.2 FRAME_END . . . 269
8.43.3.3 DELIMITER . . . 269
8.44 protocol.h . . . 269
8.45 lib/comms/receive.cpp File Reference . . . 270
8.45.1 Detailed Description . . . 271
8.45.2 Function Documentation . . . 271
8.45.2.1 on_receive() . . . 271
8.45.2.2 handle_uart_input() . . . 272
8.46 receive.cpp . . . 272
8.47 lib/comms/send.cpp File Reference . . . 273
8.47.1 Detailed Description . . . 274
8.47.2 Function Documentation . . . 274
8.47.2.1 send_message() . . . 274
8.47.2.2 send_frame_lora() . . . 275
8.47.2.3 send_frame_uart() . . . 276
8.47.2.4 send_frame() . . . 276
8.47.2.5 split_and_send_message() . . . 277
8.48 send.cpp . . . 277
8.49 lib/comms/utils_converters.cpp File Reference . . . 278

8.49.1 Detailed Description . . . 279
8.49.2 Function Documentation . . . 279
8.49.2.1 exception_type_to_string() . . . 279
8.49.2.2 value_unit_type_to_string() . . . 279
8.49.2.3 operation_type_to_string() . . . 281
8.49.2.4 string_to_operation_type() . . . 281
8.49.2.5 hex_string_to_bytes() . . . 282
8.50 utils_converters.cpp . . . 283
8.51 lib/eventman/event_manager.cpp File Reference . . . 283
8.51.1 Detailed Description . . . 284
8.51.2 Function Documentation . . . 285
8.51.2.1 check_power_events() . . . 285
8.51.3 Variable Documentation . . . 285
8.51.3.1 eventLogId . . . 285
8.51.3.2 lastPowerState . . . 286
8.51.3.3 FALL_RATE_THRESHOLD . . . 286
8.51.3.4 FALLING_TREND_REQUIRED . . . 286
8.51.3.5 VOLTAGE_LOW_THRESHOLD . . . 286
8.51.3.6 VOLTAGE_OVERCHARGE_THRESHOLD . . . 286
8.51.3.7 fallingTrendCount . . . 286
8.51.3.8 lastSolarState . . . 287
8.51.3.9 lastUSBState . . . 287
8.51.3.10 systemClock . . . 287
8.51.3.11 eventManager . . . 287
8.52 event_manager.cpp . . . 287
8.53 lib/eventman/event_manager.h File Reference . . . 289
8.53.1 Macro Definition Documentation . . . 291
8.53.1.1 EVENT_BUFFER_SIZE . . . 291
8.53.1.2 EVENT_LOG_FILE . . . 291
8.53.2 Enumeration Type Documentation . . . 291
8.53.2.1 EventGroup . . . 291
8.53.2.2 SystemEvent . . . 291
8.53.2.3 PowerEvent . . . 291
8.53.2.4 CommsEvent . . . 292
8.53.2.5 GPSEvent . . . 292
8.53.2.6 ClockEvent . . . 292
8.53.3 Function Documentation . . . 293
8.53.3.1 __attribute__() . . . 293
8.53.3.2 to_string() . . . 293
8.53.3.3 check_power_events() . . . 293
8.53.4 Variable Documentation . . . 294
8.53.4.1 id . . . 294

8.53.4.2 timestamp . . . 294
8.53.4.3 group . . . 294
8.53.4.4 event . . . 295
8.53.4.5 __attribute__ . . . 295
8.53.4.6 eventManager . . . 295
8.54 event_manager.h . . . 295
8.55 lib/location/gps_collector.cpp File Reference . . . 297
8.55.1 Macro Definition Documentation . . . 298
8.55.1.1 MAX_RAW_DATA_LENGTH . . . 298
8.55.2 Function Documentation . . . 298
8.55.2.1 splitString() . . . 298
8.55.2.2 collect_gps_data() . . . 299
8.55.3 Variable Documentation . . . 299
8.55.3.1 nmea_data . . . 299
8.56 gps_collector.cpp . . . 299
8.57 lib/location/gps_collector.h File Reference . . . 300
8.57.1 Function Documentation . . . 301
8.57.1.1 collect_gps_data() . . . 301
8.58 gps_collector.h . . . 302
8.59 lib/location/NMEA/NMEA_data.cpp File Reference . . . 302
8.59.1 Variable Documentation . . . 302
8.59.1.1 nmea_data . . . 302
8.60 NMEA_data.cpp . . . 303
8.61 lib/location/NMEA/NMEA_data.h File Reference . . . 303
8.61.1 Variable Documentation . . . 304
8.61.1.1 nmea_data . . . 304
8.62 NMEA_data.h . . . 304
8.63 lib/pin_config.cpp File Reference . . . 305
8.63.1 Variable Documentation . . . 305
8.63.1.1 lora_cs_pin . . . 305
8.63.1.2 lora_reset_pin . . . 306
8.63.1.3 lora_irq_pin . . . 306
8.63.1.4 lora_address_local . . . 306
8.63.1.5 lora_address_remote . . . 306
8.64 pin_config.cpp . . . 306
8.65 lib/pin_config.h File Reference . . . 307
8.65.1 Macro Definition Documentation . . . 308
8.65.1.1 DEBUG_UART_PORT . . . 308
8.65.1.2 DEBUG_UART_BAUD_RATE . . . 308
8.65.1.3 DEBUG_UART_TX_PIN . . . 308
8.65.1.4 DEBUG_UART_RX_PIN . . . 308
8.65.1.5 MAIN_I2C_PORT . . . 309

8.65.1.6 MAIN_I2C_SDA_PIN . . . 309
8.65.1.7 MAIN_I2C_SCL_PIN . . . 309
8.65.1.8 GPS_UART_PORT . . . 309
8.65.1.9 GPS_UART_BAUD_RATE . . . 309
8.65.1.10 GPS_UART_TX_PIN . . . 309
8.65.1.11 GPS_UART_RX_PIN . . . 309
8.65.1.12 GPS_POWER_ENABLE_PIN . . . 309
8.65.1.13 BUFFER_SIZE . . . 310
8.65.1.14 SD_SPI_PORT . . . 310
8.65.1.15 SD_MISO_PIN . . . 310
8.65.1.16 SD_MOSI_PIN . . . 310
8.65.1.17 SD_SCK_PIN . . . 310
8.65.1.18 SD_CS_PIN . . . 310
8.65.1.19 SD_CARD_DETECT_PIN . . . 310
8.65.1.20 SX1278_MISO . . . 310
8.65.1.21 SX1278_CS . . . 311
8.65.1.22 SX1278_SCK . . . 311
8.65.1.23 SX1278_MOSI . . . 311
8.65.1.24 SPI_PORT . . . 311
8.65.1.25 READ_BIT . . . 311
8.65.1.26 LORA_DEFAULT_SPI . . . 311
8.65.1.27 LORA_DEFAULT_SPI_FREQUENCY . . . 311
8.65.1.28 LORA_DEFAULT_SS_PIN . . . 311
8.65.1.29 LORA_DEFAULT_RESET_PIN . . . 312
8.65.1.30 LORA_DEFAULT_DIO0_PIN . . . 312
8.65.1.31 PA_OUTPUT_RFO_PIN . . . 312
8.65.1.32 PA_OUTPUT_PA_BOOST_PIN . . . 312
8.65.2 Variable Documentation . . . 312
8.65.2.1 lora_cs_pin . . . 312
8.65.2.2 lora_reset_pin . . . 312
8.65.2.3 lora_irq_pin . . . 312
8.65.2.4 lora_address_local . . . 313
8.65.2.5 lora_address_remote . . . 313
8.66 pin_config.h . . . 313
8.67 lib/powerman/INA3221/INA3221.cpp File Reference . . . 314
8.67.1 Detailed Description . . . 314
8.68 INA3221.cpp . . . 314
8.69 lib/powerman/INA3221/INA3221.h File Reference . . . 319
8.69.1 Detailed Description . . . 320
8.69.2 Enumeration Type Documentation . . . 320
8.69.2.1 ina3221_addr_t . . . 320
8.69.2.2 ina3221_ch_t . . . 321

8.69.2.3 ina3221_reg_t . . . 321
8.69.2.4 ina3221_conv_time_t . . . 322
8.69.2.5 ina3221_avg_mode_t . . . 322
8.69.3 Variable Documentation . . . 322
8.69.3.1 INA3221_CH_NUM . . . 322
8.69.3.2 SHUNT_VOLTAGE_LSB_UV . . . 323
8.70 INA3221.h . . . 323
8.71 lib/powerman/PowerManager.cpp File Reference . . . 325
8.72 PowerManager.cpp . . . 325
8.73 lib/powerman/PowerManager.h File Reference . . . 327
8.74 PowerManager.h . . . 328
8.75 lib/sensors/BH1750/BH1750.cpp File Reference . . . 329
8.76 BH1750.cpp . . . 329
8.77 lib/sensors/BH1750/BH1750.h File Reference . . . 330
8.77.1 Macro Definition Documentation . . . 331
8.77.1.1 _BH1750_DEVICE_ID . . . 331
8.77.1.2 _BH1750_MTREG_MIN . . . 331
8.77.1.3 _BH1750_MTREG_MAX . . . 332
8.77.1.4 _BH1750_DEFAULT_MTREG . . . 332
8.78 BH1750.h . . . 332
8.79 lib/sensors/BH1750/BH1750_WRAPPER.cpp File Reference . . . 332
8.80 BH1750_WRAPPER.cpp . . . 333
8.81 lib/sensors/BH1750/BH1750_WRAPPER.h File Reference . . . 334
8.82 BH1750_WRAPPER.h . . . 335
8.83 lib/sensors/BME280/BME280.cpp File Reference . . . 335
8.84 BME280.cpp . . . 336
8.85 lib/sensors/BME280/BME280.h File Reference . . . 339
8.86 BME280.h . . . 340
8.87 lib/sensors/BME280/BME280_WRAPPER.cpp File Reference . . . 342
8.88 BME280_WRAPPER.cpp . . . 342
8.89 lib/sensors/BME280/BME280_WRAPPER.h File Reference . . . 343
8.90 BME280_WRAPPER.h . . . 343
8.91 lib/sensors/HMC5883L/HMC5883L.cpp File Reference . . . 344
8.92 HMC5883L.cpp . . . 344
8.93 lib/sensors/HMC5883L/HMC5883L.h File Reference . . . 345
8.94 HMC5883L.h . . . 346
8.95 lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp File Reference . . . 346
8.96 HMC5883L_WRAPPER.cpp . . . 347
8.97 lib/sensors/HMC5883L/HMC5883L_WRAPPER.h File Reference . . . 348
8.98 HMC5883L_WRAPPER.h . . . 349
8.99 lib/sensors/ISensor.cpp File Reference . . . 349
8.99.1 Detailed Description . . . 350

8.100 ISensor.cpp . . . 350
8.101 lib/sensors/ISensor.h File Reference . . . 350
8.101.1 Enumeration Type Documentation . . . 351
8.101.1.1 SensorType . . . 351
8.101.1.2 SensorDataTypeIdentifier . . . 352
8.102 ISensor.h . . . 352
8.103 lib/sensors/MPU6050/MPU6050.cpp File Reference . . . 353
8.104 MPU6050.cpp . . . 353
8.105 lib/sensors/MPU6050/MPU6050.h File Reference . . . 354
8.106 MPU6050.h . . . 354
8.107 lib/sensors/MPU6050/MPU6050_WRAPPER.cpp File Reference . . . 354
8.108 MPU6050_WRAPPER.cpp . . . 354
8.109 lib/sensors/MPU6050/MPU6050_WRAPPER.h File Reference . . . 354
8.110 MPU6050_WRAPPER.h . . . 355
8.111 lib/storage/storage.cpp File Reference . . . 356
8.111.1 Detailed Description . . . 356
8.111.2 Function Documentation . . . 357
8.111.2.1 fs_init() . . . 357
8.111.3 Variable Documentation . . . 358
8.111.3.1 sd_card_mounted . . . 358
8.112 storage.cpp . . . 358
8.113 lib/storage/storage.h File Reference . . . 358
8.113.1 Function Documentation . . . 359
8.113.1.1 fs_init() . . . 359
8.113.1.2 fs_open_file() . . . 360
8.113.1.3 fs_write_file() . . . 360
8.113.1.4 fs_read_file() . . . 361
8.113.1.5 fs_close_file() . . . 361
8.113.1.6 fs_file_exists() . . . 361
8.113.2 Variable Documentation . . . 361
8.113.2.1 sd_card_mounted . . . 361
8.114 storage.h . . . 361
8.115 lib/utils.cpp File Reference . . . 362
8.115.1 Detailed Description . . . 363
8.115.2 Function Documentation . . . 363
8.115.2.1 get_level_color() . . . 363
8.115.2.2 get_level_prefix() . . . 364
8.115.2.3 uart_print() . . . 365
8.115.2.4 crc16() . . . 366
8.115.3 Variable Documentation . . . 367
8.115.3.1 uart_mutex . . . 367
8.115.3.2 g_uart_verbosity . . . 367

8.116 utils.cpp . . . 367
8.117 lib/utils.h File Reference . . . 368
8.117.1 Detailed Description . . . 370
8.117.2 Macro Definition Documentation . . . 370
8.117.2.1 ANSI_RED . . . 370
8.117.2.2 ANSI_GREEN . . . 370
8.117.2.3 ANSI_YELLOW . . . 370
8.117.2.4 ANSI_BLUE . . . 370
8.117.2.5 ANSI_CYAN . . . 370
8.117.2.6 ANSI_RESET . . . 370
8.117.3 Enumeration Type Documentation . . . 370
8.117.3.1 VerbosityLevel . . . 370
8.117.4 Function Documentation . . . 371
8.117.4.1 uart_print() . . . 371
8.117.4.2 crc16() . . . 372
8.117.5 Variable Documentation . . . 373
8.117.5.1 g_uart_verbosity . . . 373
8.118 utils.h . . . 374
8.119 main.cpp File Reference . . . 374
8.119.1 Macro Definition Documentation . . . 375
8.119.1.1 LOG_FILENAME . . . 375
8.119.2 Function Documentation . . . 375
8.119.2.1 core1_entry() . . . 375
8.119.2.2 init_systems() . . . 376
8.119.2.3 main() . . . 376
8.119.3 Variable Documentation . . . 377
8.119.3.1 powerManager . . . 377
8.119.3.2 systemClock . . . 377
8.119.3.3 buffer . . . 377
8.119.3.4 bufferIndex . . . 377
8.120 main.cpp . . . 378
8.121 test/comms/test_converters.cpp File Reference . . . 379
8.121.1 Function Documentation . . . 380
8.121.1.1 test_operation_type_conversion() . . . 380
8.121.1.2 test_value_unit_type_conversion() . . . 381
8.121.1.3 test_exception_type_conversion() . . . 381
8.121.1.4 test_hex_string_conversion() . . . 382
8.122 test_converters.cpp . . . 382
8.123 test/comms/test_frame_build.cpp File Reference . . . 383
8.123.1 Function Documentation . . . 383
8.123.1.1 test_frame_build_success() . . . 383
8.123.1.2 test_frame_build_error() . . . 384

8.123.1.3 test_frame_build_info() . . . 385
8.124 test_frame_build.cpp . . . 385
8.125 test/comms/test_frame_coding.cpp File Reference . . . 386
8.125.1 Function Documentation . . . 386
8.125.1.1 test_frame_encode_basic() . . . 386
8.125.1.2 test_frame_decode_basic() . . . 387
8.125.1.3 test_frame_decode_invalid_header() . . . 387
8.126 test_frame_coding.cpp . . . 388
8.127 test/comms/test_frame_common.h File Reference . . . 388
8.127.1 Function Documentation . . . 389
8.127.1.1 create_test_frame() . . . 389
8.128 test_frame_common.h . . . 390
8.129 test/test_runner.cpp File Reference . . . 390
8.129.1 Function Documentation . . . 391
8.129.1.1 test_frame_encode_basic() . . . 391
8.129.1.2 test_frame_decode_basic() . . . 391
8.129.1.3 test_frame_decode_invalid_header() . . . 392
8.129.1.4 test_frame_build_success() . . . 392
8.129.1.5 test_frame_build_error() . . . 393
8.129.1.6 test_frame_build_info() . . . 394
8.129.1.7 test_operation_type_conversion() . . . 394
8.129.1.8 test_value_unit_type_conversion() . . . 395
8.129.1.9 test_exception_type_conversion() . . . 396
8.129.1.10 test_hex_string_conversion() . . . 396
8.129.1.11 main() . . . 397
8.130 test_runner.cpp . . . 397

Index **399**

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_file_download (const std::string &param, OperationType operationType)

Command ID: 6.1

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

Command ID: 1

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

Command ID: 0

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

Command ID: 2.5

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

Command ID: 2.6

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

Command ID: 2.4

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

Command ID: 2.7

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

Command ID: 5.2

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

Command ID: 7.4

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

Command ID: 5.1

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

Command ID: 3.7

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

Command ID: 2.0

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

Command ID: 7.3

Member handle_get_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_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 14
Diagnostic Commands 17
Event Commands 21
GPS Commands 23
Power Commands 27
Storage Commands 33
INA3221 Power Monitor 36
- Configuration Functions 36
- Measurement Functions 46
- Alert Functions 48

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

BH1750 55
BME280 61
BME280CalibParam 70
INA3221::conf_reg_t 76
DS3231 78
ds3231_data_t 85
EventEmitter 87
EventLog 88
EventManager 90
    EventManagerImpl 96
FileHandle 99
Frame 99
HMC5883L 101
INA3221 106
ISensor 112
    BH1750Wrapper 58
    BME280Wrapper 73
    HMC5883LWrapper 104
    MPU6050Wrapper 145
INA3221::masken_reg_t 142
NMEAData 148
PowerManager 150
Print 156
    LoRaClass 114
SensorWrapper 176

Chapter 4

Class Index

4.1 Class List

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

BH1750 55
BH1750Wrapper 58
BME280 61
BME280CalibParam 70
BME280Wrapper 73
INA3221::conf_reg_t
Configuration register bit fields 76
DS3231 78
ds3231_data_t 85
EventEmitter
Provides a static method for emitting events 87
EventLog
Represents a single event log entry 88
EventManager
Manages the event logging system 90
EventManagerImpl
Implementation of the EventManager class 96
FileHandle 99
Frame 99
HMC5883L 101
HMC5883LWrapper 104
INA3221
INA3221 Triple-Channel Power Monitor driver class 106
ISensor 112
LoRaClass 114
INA3221::masken_reg_t
Mask/Enable register bit fields 142
MPU6050Wrapper 145
NMEAData 148
PowerManager 150
Print 156
SensorWrapper
Manages different sensor types and provides a unified interface for accessing sensor data . . 176

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 . . . 374
lib/pin_config.cpp . . . 305
lib/pin_config.h . . . 307
lib/utils.cpp
Implementation of utility functions for the Kubisat firmware . . . 362
lib/utils.h
Utility functions and definitions for the Kubisat firmware . . . 368
lib/clock/DS3231.cpp . . . 183
lib/clock/DS3231.h . . . 187
lib/comms/communication.cpp . . . 213
lib/comms/communication.h . . . 216
lib/comms/frame.cpp
Implements functions for encoding, decoding, building, and processing Frames . . . 227
lib/comms/protocol.h . . . 261
lib/comms/receive.cpp
Implements functions for receiving and processing data, including LoRa and UART input . . . 270
lib/comms/send.cpp
Implements functions for sending data, including LoRa messages and Frames . . . 273
lib/comms/utils_converters.cpp
Implements utility functions for converting between different data types . . . 278
lib/comms/commands/clock_commands.cpp . . . 190
lib/comms/commands/commands.cpp . . . 194
lib/comms/commands/commands.h . . . 195
lib/comms/commands/diagnostic_commands.cpp . . . 198
lib/comms/commands/event_commands.cpp . . . 199
lib/comms/commands/gps_commands.cpp . . . 201
lib/comms/commands/power_commands.cpp . . . 203
lib/comms/commands/storage_commands.cpp . . . 205
lib/comms/commands/storage_commands_utils.cpp . . . 209
lib/comms/commands/storage_commands_utils.h . . . 211
lib/comms/LoRa/LoRa-RP2040.cpp . . . 234
lib/comms/LoRa/LoRa-RP2040.h . . . 251
lib/comms/LoRa/Print.cpp . . . 254

lib/comms/LoRa/Print.h . . . 259
lib/eventman/event_manager.cpp
Implements the event management system for the Kubisat firmware . . . 283
lib/eventman/event_manager.h . . . 289
lib/location/gps_collector.cpp . . . 297
lib/location/gps_collector.h . . . 300
lib/location/NMEA/NMEA_data.cpp . . . 302
lib/location/NMEA/NMEA_data.h . . . 303
lib/powerman/PowerManager.cpp . . . 325
lib/powerman/PowerManager.h . . . 327
lib/powerman/INA3221/INA3221.cpp
Implementation of the INA3221 power monitor driver . . . 314
lib/powerman/INA3221/INA3221.h
Header file for the INA3221 triple-channel power monitor driver . . . 319
lib/sensors/ISensor.cpp
Implements the SensorWrapper class for managing different sensor types . . . 349
lib/sensors/ISensor.h . . . 350
lib/sensors/BH1750/BH1750.cpp . . . 329
lib/sensors/BH1750/BH1750.h . . . 330
lib/sensors/BH1750/BH1750_WRAPPER.cpp . . . 332
lib/sensors/BH1750/BH1750_WRAPPER.h . . . 334
lib/sensors/BME280/BME280.cpp . . . 335
lib/sensors/BME280/BME280.h . . . 339
lib/sensors/BME280/BME280_WRAPPER.cpp . . . 342
lib/sensors/BME280/BME280_WRAPPER.h . . . 343
lib/sensors/HMC5883L/HMC5883L.cpp . . . 344
lib/sensors/HMC5883L/HMC5883L.h . . . 345
lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp . . . 346
lib/sensors/HMC5883L/HMC5883L_WRAPPER.h . . . 348
lib/sensors/MPU6050/MPU6050.cpp . . . 353
lib/sensors/MPU6050/MPU6050.h . . . 354
lib/sensors/MPU6050/MPU6050_WRAPPER.cpp . . . 354
lib/sensors/MPU6050/MPU6050_WRAPPER.h . . . 354
lib/storage/storage.cpp
Implements file system operations for the Kubisat firmware . . . 356
lib/storage/storage.h . . . 358
test/test_runner.cpp . . . 390
test/comms/test_converters.cpp . . . 379
test/comms/test_frame_build.cpp . . . 383
test/comms/test_frame_coding.cpp . . . 386
test/comms/test_frame_common.h . . . 388

Chapter 6

Topic Documentation

6.1 Clock Management Commands

Commands for managing system time and clock settings.

Functions

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

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

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

  Handler for getting and setting clock synchronization interval.
- 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()

```
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

Frame 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()

```
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

Frame 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 71 of file clock_commands.cpp.

Here is the call graph for this function:

6.1.2.3 handle_clock_sync_interval()

```
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

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 115 of file clock_commands.cpp.

Here is the call graph for this function:

6.1.2.4 handle_get_last_sync_time()

```
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

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 155 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<Frame(const std::string&, OperationType)>

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

  Map type for storing command handlers.

Functions

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

  Executes a command based on its key.

Variables

- CommandMap commandHandlers

  Global map of all command handlers.

6.2.1 Detailed Description

Core command system implementation.

6.2.2 Typedef Documentation

6.2.2.1 CommandHandler

```
using CommandHandler = std::function<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()

```
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 63 of file commands.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

6.2.4 Variable Documentation

6.2.4.1 commandHandlers

```
CommandMap commandHandlers
```

Initial value:

```
= {
    {((static_cast<uint32_t>(1) « 8) | static_cast<uint32_t>(0)), handle_get_commands_list},
    {((static_cast<uint32_t>(1) « 8) | static_cast<uint32_t>(1)), handle_get_build_version},
    {((static_cast<uint32_t>(1) « 8) | static_cast<uint32_t>(8)), handle_verbosity},
    {((static_cast<uint32_t>(1) « 8) | static_cast<uint32_t>(9)), handle_enter_bootloader_mode},
    {((static_cast<uint32_t>(2) « 8) | static_cast<uint32_t>(0)), handle_get_power_manager_ids},
    {((static_cast<uint32_t>(2) « 8) | static_cast<uint32_t>(2)), handle_get_voltage_battery},
    {((static_cast<uint32_t>(2) « 8) | static_cast<uint32_t>(3)), handle_get_voltage_5v},
    {((static_cast<uint32_t>(2) « 8) | static_cast<uint32_t>(4)), handle_get_current_charge_usb},
    {((static_cast<uint32_t>(2) « 8) | static_cast<uint32_t>(5)), handle_get_current_charge_solar},
    {((static_cast<uint32_t>(2) « 8) | static_cast<uint32_t>(6)), handle_get_current_charge_total},
```

```
    {((static_cast<uint32_t>(2) « 8) | static_cast<uint32_t>(7)), handle_get_current_draw},
    {((static_cast<uint32_t>(3) « 8) | static_cast<uint32_t>(0)), handle_time},
    {((static_cast<uint32_t>(3) « 8) | static_cast<uint32_t>(1)), handle_timezone_offset},
    {((static_cast<uint32_t>(3) « 8) | static_cast<uint32_t>(2)), handle_clock_sync_interval},
    {((static_cast<uint32_t>(3) « 8) | static_cast<uint32_t>(3)), handle_get_last_sync_time},
    {((static_cast<uint32_t>(5) « 8) | static_cast<uint32_t>(1)), handle_get_last_events},
    {((static_cast<uint32_t>(5) « 8) | static_cast<uint32_t>(2)), handle_get_event_count},
    {((static_cast<uint32_t>(6) « 8) | static_cast<uint32_t>(0)), handle_list_files},
    {((static_cast<uint32_t>(6) « 8) | static_cast<uint32_t>(1)), handle_file_download},
    {((static_cast<uint32_t>(6) « 8) | static_cast<uint32_t>(4)), handle_mount},
    {((static_cast<uint32_t>(7) « 8) | static_cast<uint32_t>(1)), handle_gps_power_status},
    {((static_cast<uint32_t>(7) « 8) | static_cast<uint32_t>(2)), handle_enable_gps_uart_passthrough},
    {((static_cast<uint32_t>(7) « 8) | static_cast<uint32_t>(3)), handle_get_rmc_data},
    {((static_cast<uint32_t>(7) « 8) | static_cast<uint32_t>(4)), handle_get_gga_data},
}
```

Global map of all command handlers.

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

Definition at line 27 of file commands.cpp.

6.3 Diagnostic Commands

Functions

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

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

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

  Handles setting or getting the UART verbosity level.
- Frame handle_enter_bootloader_mode (const std::string &param, OperationType operationType)

  Reboot system to USB firmware loader.

6.3.1 Detailed Description

6.3.2 Function Documentation

6.3.2.1 handle_get_commands_list()

```
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

Frame containing success/error and command list

Note

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

Print all available commands on UART port

Command Command ID: 0

Definition at line 21 of file diagnostic_commands.cpp.

Here is the call graph for this function:

6.3.2.2 handle_get_build_version()

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

Get firmware build version.

Parameters

param	Empty string expected
operationType	GET

Returns

Frame containing build number

Note

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

Get the firmware build version

Command Command ID: 1

Definition at line 56 of file diagnostic_commands.cpp.

Here is the call graph for this function:

6.3.2.3 handle_verbosity()

```
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

A 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 88 of file diagnostic_commands.cpp.

Here is the call graph for this function:

6.3.2.4 handle_enter_bootloader_mode()

```
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 117 of file diagnostic_commands.cpp.

Here is the call graph for this function:

6.4 Event Commands

Commands for accessing and managing system event logs.

Functions

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

  Handler for retrieving last N events from the event log.
- 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()

```
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)
operationType	GET

Returns

Frame containing:

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

Note

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

Returns up to 10 most recent events

Command Command ID: 5.1

Definition at line 29 of file event_commands.cpp.

Here is the call graph for this function:

6.4.2.2 handle_get_event_count()

```
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 83 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

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

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

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

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

```
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

Frame containing:

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

Note

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

Return current GPS module power state: ON/OFF

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

POWER - 0 - OFF, 1 - ON

Command Command ID: 7.1

Definition at line 26 of file gps_commands.cpp.

Here is the call graph for this function:

6.5.2.2 handle_enable_gps_uart_passthrough()

```
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

Frame 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 74 of file gps_commands.cpp.

Here is the call graph for this function:

6.5.2.3 handle_get_rmc_data()

```
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

Frame 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 170 of file gps_commands.cpp.

Here is the call graph for this function:

6.5.2.4 handle_get_gga_data()

```
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

Frame 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 209 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

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

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

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

  Handler for getting 5V rail voltage.
- Frame handle_get_current_charge_usb (const std::string &param, OperationType operationType)

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

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

  Handler for getting total charge current.
- 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()

```
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

Frame 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 22 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.2 handle_get_voltage_battery()

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

Handler for getting battery voltage.

Parameters

param	Empty string expected
operationType	GET

Returns

Frame 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 49 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.3 handle_get_voltage_5v()

```
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

Frame 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 76 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.4 handle_get_current_charge_usb()

```
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

Frame 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 103 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.5 handle_get_current_charge_solar()

```
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

Frame 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 130 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.6 handle_get_current_charge_total()

```
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

Frame 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 157 of file power_commands.cpp.

Here is the call graph for this function:

6.6.2.7 handle_get_current_draw()

```
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

Frame 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 184 of file power_commands.cpp.

Here is the call graph for this function:

6.7 Storage Commands

Commands for interacting with the SD card storage.

Functions

- Frame handle_file_download (const std::string &param, OperationType operationType)

  Handles the file download command.
- Frame handle_list_files (const std::string &param, OperationType operationType)

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

  Handles the SD card mount/unmount command.

6.7.1 Detailed Description

Commands for interacting with the SD card storage.

6.7.2 Function Documentation

6.7.2.1 handle_file_download()

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

Handles the file download command.

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

Parameters

param	The filename to download.
operationType	The operation type (must be GET).

Returns

A Frame indicating the result of the operation.

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

Note

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

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

Command Command ID: 6.1

Definition at line 43 of file storage_commands.cpp.

Here is the call graph for this function:

6.7.2.2 handle_list_files()

```
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 in separate frames.

Parameters

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

Returns

A Frame 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 120 of file storage_commands.cpp.

Here is the call graph for this function:

6.7.2.3 handle_mount()

```
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 Frame 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 180 of file storage_commands.cpp.

Here is the call graph for this function:

6.8 INA3221 Power Monitor

Collaboration diagram for INA3221 Power Monitor:

Topics

- Configuration Functions
- Measurement Functions
- Alert Functions

6.8.1 Detailed Description

6.8.2 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.

6.8.2.1 Detailed Description

Functions for configuring the INA3221 device

6.8.2.2 Function Documentation

6.8.2.2.1 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.

6.8.2.2.2 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:

6.8.2.2.3 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:

6.8.2.2.4 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:

6.8.2.2.5 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:

6.8.2.2.6 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:

6.8.2.2.7 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:

6.8.2.2.8 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:

6.8.2.2.9 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:

6.8.2.2.10 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:

6.8.2.2.11 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:

6.8.2.2.12 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:

6.8.2.2.13 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:

6.8.2.2.14 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:

6.8.2.2.15 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:

6.8.2.2.16 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:

6.8.3 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.

6.8.3.1 Detailed Description

Functions for reading voltage, current and power measurements

6.8.3.2 Function Documentation

6.8.3.2.1 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:

6.8.3.2.2 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:

6.8.3.2.3 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:

6.8.4 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.

6.8.4.1 Detailed Description

Functions for configuring and reading alert conditions

6.8.4.2 Function Documentation

6.8.4.2.1 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:

6.8.4.2.2 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:

6.8.4.2.3 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:

6.8.4.2.4 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:

6.8.4.2.5 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:

6.8.4.2.6 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:

6.8.4.2.7 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:

6.8.4.2.8 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:

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)

Public Member Functions inherited from ISensor

- virtual ~ISensor ()=default

Private Attributes

- BH1750 sensor_
- bool initialized_ = false

7.2.1 Detailed Description

Definition at line 9 of file BH1750_WRAPPER.h.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 BH1750Wrapper()

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

Definition at line 6 of file BH1750_WRAPPER.cpp.

7.2.3 Member Function Documentation

7.2.3.1 get_i2c_addr()

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

7.2.3.2 init()

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

Implements ISensor.

Definition at line 10 of file BH1750_WRAPPER.cpp.

7.2.3.3 read_data()

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

Implements ISensor.

Definition at line 15 of file BH1750_WRAPPER.cpp.

7.2.3.4 is_initialized()

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

Implements ISensor.

Definition at line 22 of file BH1750_WRAPPER.cpp.

7.2.3.5 get_type()

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

Implements ISensor.

Definition at line 26 of file BH1750_WRAPPER.cpp.

7.2.3.6 configure()

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

Implements ISensor.

Definition at line 30 of file BH1750_WRAPPER.cpp.

7.2.4 Member Data Documentation

7.2.4.1 sensor_

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

Definition at line 11 of file BH1750_WRAPPER.h.

7.2.4.2 initialized_

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

Definition at line 12 of file BH1750_WRAPPER.h.

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

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

7.3 BME280 Class Reference

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

Collaboration diagram for BME280:

Public Member Functions

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

Static Public Attributes

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

Private Member Functions

- bool configure_sensor ()
- bool get_calibration_parameters ()

Private Attributes

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

Static Private Attributes

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

7.3.1 Detailed Description

Definition at line 38 of file BME280.h.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 BME280()

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

Definition at line 14 of file BME280.cpp.

7.3.3 Member Function Documentation

7.3.3.1 init()

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

Definition at line 18 of file BME280.cpp.

Here is the call graph for this function:

7.3.3.2 reset()

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

Definition at line 59 of file BME280.cpp.

7.3.3.3 read_raw_all()

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

Definition at line 68 of file BME280.cpp.

7.3.3.4 convert_temperature()

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

Definition at line 101 of file BME280.cpp.

7.3.3.5 convert_pressure()

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

Definition at line 110 of file BME280.cpp.

7.3.3.6 convert_humidity()

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

Definition at line 131 of file BME280.cpp.

7.3.3.7 configure_sensor()

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

Definition at line 201 of file BME280.cpp.

Here is the caller graph for this function:

7.3.3.8 get_calibration_parameters()

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

Definition at line 143 of file BME280.cpp.

Here is the caller graph for this function:

7.3.4 Member Data Documentation

7.3.4.1 ADDR_SDO_LOW

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

Definition at line 41 of file BME280.h.

7.3.4.2 ADDR_SDO_HIGH

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

Definition at line 42 of file BME280.h.

7.3.4.3 i2c_port

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

Definition at line 69 of file BME280.h.

7.3.4.4 device_addr

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

Definition at line 70 of file BME280.h.

7.3.4.5 calib_params

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

Definition at line 73 of file BME280.h.

7.3.4.6 initialized_

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

Definition at line 76 of file BME280.h.

7.3.4.7 t_fine

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

Definition at line 79 of file BME280.h.

7.3.4.8 REG_CONFIG

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

Definition at line 82 of file BME280.h.

7.3.4.9 REG_CTRL_MEAS

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

Definition at line 83 of file BME280.h.

7.3.4.10 REG_CTRL_HUM

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

Definition at line 84 of file BME280.h.

7.3.4.11 REG_RESET

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

Definition at line 85 of file BME280.h.

7.3.4.12 REG_PRESSURE_MSB

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

Definition at line 87 of file BME280.h.

7.3.4.13 REG_TEMPERATURE_MSB

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

Definition at line 88 of file BME280.h.

7.3.4.14 REG_HUMIDITY_MSB

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

Definition at line 89 of file BME280.h.

7.3.4.15 REG_DIG_T1_LSB

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

Definition at line 92 of file BME280.h.

7.3.4.16 REG_DIG_T1_MSB

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

Definition at line 93 of file BME280.h.

7.3.4.17 REG_DIG_T2_LSB

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

Definition at line 94 of file BME280.h.

7.3.4.18 REG_DIG_T2_MSB

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

Definition at line 95 of file BME280.h.

7.3.4.19 REG_DIG_T3_LSB

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

Definition at line 96 of file BME280.h.

7.3.4.20 REG_DIG_T3_MSB

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

Definition at line 97 of file BME280.h.

7.3.4.21 REG_DIG_P1_LSB

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

Definition at line 99 of file BME280.h.

7.3.4.22 REG_DIG_P1_MSB

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

Definition at line 100 of file BME280.h.

7.3.4.23 REG_DIG_P2_LSB

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

Definition at line 101 of file BME280.h.

7.3.4.24 REG_DIG_P2_MSB

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

Definition at line 102 of file BME280.h.

7.3.4.25 REG_DIG_P3_LSB

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

Definition at line 103 of file BME280.h.

7.3.4.26 REG_DIG_P3_MSB

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

Definition at line 104 of file BME280.h.

7.3.4.27 REG_DIG_P4_LSB

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

Definition at line 105 of file BME280.h.

7.3.4.28 REG_DIG_P4_MSB

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

Definition at line 106 of file BME280.h.

7.3.4.29 REG_DIG_P5_LSB

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

Definition at line 107 of file BME280.h.

7.3.4.30 REG_DIG_P5_MSB

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

Definition at line 108 of file BME280.h.

7.3.4.31 REG_DIG_P6_LSB

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

Definition at line 109 of file BME280.h.

7.3.4.32 REG_DIG_P6_MSB

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

Definition at line 110 of file BME280.h.

7.3.4.33 REG_DIG_P7_LSB

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

Definition at line 111 of file BME280.h.

7.3.4.34 REG_DIG_P7_MSB

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

Definition at line 112 of file BME280.h.

7.3.4.35 REG_DIG_P8_LSB

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

Definition at line 113 of file BME280.h.

7.3.4.36 REG_DIG_P8_MSB

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

Definition at line 114 of file BME280.h.

7.3.4.37 REG_DIG_P9_LSB

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

Definition at line 115 of file BME280.h.

7.3.4.38 REG_DIG_P9_MSB

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

Definition at line 116 of file BME280.h.

7.3.4.39 REG_DIG_H1

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

Definition at line 119 of file BME280.h.

7.3.4.40 REG_DIG_H2

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

Definition at line 120 of file BME280.h.

7.3.4.41 REG_DIG_H3

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

Definition at line 121 of file BME280.h.

7.3.4.42 REG_DIG_H4

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

Definition at line 122 of file BME280.h.

7.3.4.43 REG_DIG_H5

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

Definition at line 123 of file BME280.h.

7.3.4.44 REG_DIG_H6

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

Definition at line 124 of file BME280.h.

7.3.4.45 NUM_CALIB_PARAMS

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

Definition at line 127 of file BME280.h.

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

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

7.4 BME280CalibParam Struct Reference

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

Public Attributes

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

7.4.1 Detailed Description

Definition at line 11 of file BME280.h.

7.4.2 Member Data Documentation

7.4.2.1 dig_t1

```
uint16_t BME280CalibParam::dig_t1
```

Definition at line 13 of file BME280.h.

7.4.2.2 dig_t2

```
int16_t BME280CalibParam::dig_t2
```

Definition at line 14 of file BME280.h.

7.4.2.3 dig_t3

```
int16_t BME280CalibParam::dig_t3
```

Definition at line 15 of file BME280.h.

7.4.2.4 dig_p1

```
uint16_t BME280CalibParam::dig_p1
```

Definition at line 18 of file BME280.h.

7.4.2.5 dig_p2

```
int16_t BME280CalibParam::dig_p2
```

Definition at line 19 of file BME280.h.

7.4.2.6 dig_p3

```
int16_t BME280CalibParam::dig_p3
```

Definition at line 20 of file BME280.h.

7.4.2.7 dig_p4

```
int16_t BME280CalibParam::dig_p4
```

Definition at line 21 of file BME280.h.

7.4.2.8 dig_p5

```
int16_t BME280CalibParam::dig_p5
```

Definition at line 22 of file BME280.h.

7.4.2.9 dig_p6

```
int16_t BME280CalibParam::dig_p6
```

Definition at line 23 of file BME280.h.

7.4.2.10 dig_p7

```
int16_t BME280CalibParam::dig_p7
```

Definition at line 24 of file BME280.h.

7.4.2.11 dig_p8

```
int16_t BME280CalibParam::dig_p8
```

Definition at line 25 of file BME280.h.

7.4.2.12 dig_p9

```
int16_t BME280CalibParam::dig_p9
```

Definition at line 26 of file BME280.h.

7.4.2.13 dig_h1

```
uint8_t BME280CalibParam::dig_h1
```

Definition at line 29 of file BME280.h.

7.4.2.14 dig_h2

```
int16_t BME280CalibParam::dig_h2
```

Definition at line 30 of file BME280.h.

7.4.2.15 dig_h3

```
uint8_t BME280CalibParam::dig_h3
```

Definition at line 31 of file BME280.h.

7.4.2.16 dig_h4

```
int16_t BME280CalibParam::dig_h4
```

Definition at line 32 of file BME280.h.

7.4.2.17 dig_h5

```
int16_t BME280CalibParam::dig_h5
```

Definition at line 33 of file BME280.h.

7.4.2.18 dig_h6

```
int8_t BME280CalibParam::dig_h6
```

Definition at line 34 of file BME280.h.

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

- lib/sensors/BME280/BME280.h

7.5 BME280Wrapper Class Reference

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

Inheritance diagram for BME280Wrapper:

Collaboration diagram for BME280Wrapper:

Public Member Functions

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

Public Member Functions inherited from ISensor

- virtual ~ISensor ()=default

Private Attributes

- BME280 sensor_
- bool initialized_ = false

7.5.1 Detailed Description

Definition at line 8 of file BME280_WRAPPER.h.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 BME280Wrapper()

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

Definition at line 3 of file BME280_WRAPPER.cpp.

7.5.3 Member Function Documentation

7.5.3.1 init()

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

Implements ISensor.

Definition at line 5 of file BME280_WRAPPER.cpp.

7.5.3.2 read_data()

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

Implements ISensor.

Definition at line 10 of file BME280_WRAPPER.cpp.

7.5.3.3 is_initialized()

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

Implements ISensor.

Definition at line 26 of file BME280_WRAPPER.cpp.

7.5.3.4 get_type()

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

Implements ISensor.

Definition at line 30 of file BME280_WRAPPER.cpp.

7.5.3.5 configure()

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

Implements ISensor.

Definition at line 34 of file BME280_WRAPPER.cpp.

7.5.4 Member Data Documentation

7.5.4.1 sensor_

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

Definition at line 10 of file BME280_WRAPPER.h.

7.5.4.2 initialized_

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

Definition at line 11 of file BME280_WRAPPER.h.

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

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

7.6 INA3221::conf_reg_t Struct Reference

Configuration register bit fields.

Public Attributes

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

7.6.1 Detailed Description

Configuration register bit fields.

Definition at line 101 of file INA3221.h.

7.6.2 Member Data Documentation

7.6.2.1 mode_shunt_en

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

Definition at line 102 of file INA3221.h.

7.6.2.2 mode_bus_en

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

Definition at line 103 of file INA3221.h.

7.6.2.3 mode_continious_en

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

Definition at line 104 of file INA3221.h.

7.6.2.4 shunt_conv_time

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

Definition at line 105 of file INA3221.h.

7.6.2.5 bus_conv_time

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

Definition at line 106 of file INA3221.h.

7.6.2.6 avg_mode

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

Definition at line 107 of file INA3221.h.

7.6.2.7 ch3_en

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

Definition at line 108 of file INA3221.h.

7.6.2.8 ch2_en

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

Definition at line 109 of file INA3221.h.

7.6.2.9 ch1_en

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

Definition at line 110 of file INA3221.h.

7.6.2.10 reset

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

Definition at line 111 of file INA3221.h.

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

- lib/powerman/INA3221/INA3221.h

7.7 DS3231 Class Reference

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

Public Member Functions

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

Private Member Functions

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

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

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

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

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

Private Attributes

- i2c_inst_t * i2c
- uint8_t ds3231_addr
- recursive_mutex_t clock_mutex_

7.7.1 Detailed Description

Definition at line 48 of file DS3231.h.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 DS3231()

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

Definition at line 6 of file DS3231.cpp.

7.7.3 Member Function Documentation

7.7.3.1 set_time()

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

Definition at line 11 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.2 get_time()

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

Definition at line 68 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.3 read_temperature()

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

Definition at line 110 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.4 set_unix_time()

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

Definition at line 217 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.5 get_unix_time()

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

Definition at line 237 of file DS3231.cpp.

Here is the call graph for this function:

7.7.3.6 clock_enable()

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

Definition at line 265 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.7 i2c_read_reg()

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

Library function to read a specific I2C register address.

Parameters

in	*reg_addr*	Register address to be read.
in	*length*	length of the data in bytes to be read.
out	*data*	Buffer to store the read data.

Returns

0 if successful, -1 if i2c failure.

Definition at line 136 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.8 i2c_write_reg()

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

Library function to write to a specific I2C register address.

Parameters

in	*reg_addr*	Register address to be written.
in	*length*	Length of the data to be written in bytes.
in	*data*	Pointer to the data buffer.

Returns

0 if successful, -1 if i2c failure.

Definition at line 171 of file DS3231.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.7.3.9 bin_to_bcd()

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

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

Parameters

in	*data*	Number to be converted.

Returns

Number in BCD form.

Definition at line 199 of file DS3231.cpp.

Here is the caller graph for this function:

7.7.3.10 bcd_to_bin()

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

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

Parameters

in	*bcd*	BCD number to be converted.

Returns

Unsigned 8-bit integer.

Definition at line 211 of file DS3231.cpp.

Here is the caller graph for this function:

7.7.4 Member Data Documentation

7.7.4.1 i2c

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

Definition at line 61 of file DS3231.h.

7.7.4.2 ds3231_addr

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

Definition at line 62 of file DS3231.h.

7.7.4.3 clock_mutex_

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

Definition at line 70 of file DS3231.h.

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

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

7.8 ds3231_data_t Struct Reference

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

Public Attributes

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

7.8.1 Detailed Description

Definition at line 37 of file DS3231.h.

7.8.2 Member Data Documentation

7.8.2.1 seconds

```
uint8_t ds3231_data_t::seconds
```

Definition at line 38 of file DS3231.h.

7.8.2.2 minutes

```
uint8_t ds3231_data_t::minutes
```

Definition at line 39 of file DS3231.h.

7.8.2.3 hours

```
uint8_t ds3231_data_t::hours
```

Definition at line 40 of file DS3231.h.

7.8.2.4 day

```
uint8_t ds3231_data_t::day
```

Definition at line 41 of file DS3231.h.

7.8.2.5 date

```
uint8_t ds3231_data_t::date
```

Definition at line 42 of file DS3231.h.

7.8.2.6 month

```
uint8_t ds3231_data_t::month
```

Definition at line 43 of file DS3231.h.

7.8.2.7 year

```
uint8_t ds3231_data_t::year
```

Definition at line 44 of file DS3231.h.

7.8.2.8 century

```
bool ds3231_data_t::century
```

Definition at line 45 of file DS3231.h.

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

- lib/clock/DS3231.h

7.9 EventEmitter Class Reference

Provides a static method for emitting events.

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

Static Public Member Functions

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

  Emits an event.

7.9.1 Detailed Description

Provides a static method for emitting events.

Definition at line 236 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 245 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 76 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 87 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 78 of file event_manager.h.

7.10.3.2 timestamp

```
uint32_t EventLog::timestamp
```

Unix timestamp or system time.

Definition at line 79 of file event_manager.h.

7.10.3.3 group

```
uint8_t EventLog::group
```

Event group identifier.

Definition at line 80 of file event_manager.h.

7.10.3.4 event

```
uint8_t EventLog::event
```

Specific event identifier.

Definition at line 81 of file event_manager.h.

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

- lib/eventman/event_manager.h

7.11 EventManager Class Reference

Manages the event logging system.

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

Inheritance diagram for EventManager:

Collaboration diagram for EventManager:

Public Member Functions

- EventManager ()

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

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

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

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

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

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

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

  Loads the events from storage.

Protected Attributes

- EventLog events [EVENT_BUFFER_SIZE]

  Event buffer.
- size_t eventCount

  Number of events in the buffer.
- size_t writeIndex

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

  Mutex for protecting the event buffer.
- volatile uint16_t nextEventId

  Next event ID.
- bool needsPersistence

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

7.11.1 Detailed Description

Manages the event logging system.

Definition at line 101 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 107 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 125 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 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 80 of file event_manager.cpp.

Here is the call graph for this function:

7.11.3.3 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 114 of file event_manager.cpp.

7.11.3.4 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 147 of file event_manager.h.

7.11.3.5 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:

7.11.3.6 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 162 of file event_manager.h.

7.11.4.2 eventCount

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

Number of events in the buffer.

Definition at line 163 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 164 of file event_manager.h.

7.11.4.4 eventMutex

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

Mutex for protecting the event buffer.

Definition at line 165 of file event_manager.h.

7.11.4.5 nextEventId

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

Next event ID.

Definition at line 166 of file event_manager.h.

7.11.4.6 needsPersistence

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

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

Definition at line 167 of file event_manager.h.

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

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

7.12 EventManagerImpl Class Reference

Implementation of the EventManager class.

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

Inheritance diagram for EventManagerImpl:

Collaboration diagram for EventManagerImpl:

Public Member Functions

- EventManagerImpl ()

  Constructor for the EventManagerImpl.
- bool save_to_storage () override

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

  Loads the events from storage.

Public Member Functions inherited from EventManager

- EventManager ()

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

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

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

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

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

  Gets the number of events in the buffer.

Additional Inherited Members

Protected Attributes inherited from EventManager

- EventLog events [EVENT_BUFFER_SIZE]

  Event buffer.
- size_t eventCount

  Number of events in the buffer.
- size_t writeIndex

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

  Mutex for protecting the event buffer.
- volatile uint16_t nextEventId

  Next event ID.
- bool needsPersistence

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

7.12.1 Detailed Description

Implementation of the EventManager class.

Definition at line 175 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 181 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 190 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 220 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

```
#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

Definition at line 70 of file protocol.h.

7.14.2 Member Data Documentation

7.14.2.1 header

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

Definition at line 71 of file protocol.h.

7.14.2.2 direction

```
uint8_t Frame::direction
```

Definition at line 72 of file protocol.h.

7.14.2.3 operationType

```
OperationType Frame::operationType
```

Definition at line 73 of file protocol.h.

7.14.2.4 group

```
uint8_t Frame::group
```

Definition at line 74 of file protocol.h.

7.14.2.5 command

```
uint8_t Frame::command
```

Definition at line 75 of file protocol.h.

7.14.2.6 value

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

Definition at line 76 of file protocol.h.

7.14.2.7 unit

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

Definition at line 77 of file protocol.h.

7.14.2.8 footer

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

Definition at line 78 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

Public Member Functions inherited from ISensor

- virtual ~ISensor ()=default

Private Attributes

- HMC5883L sensor_
- bool initialized_

7.16.1 Detailed Description

Definition at line 7 of file HMC5883L_WRAPPER.h.

7.16.2 Constructor & Destructor Documentation

7.16.2.1 HMC5883LWrapper()

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

Definition at line 5 of file HMC5883L_WRAPPER.cpp.

7.16.3 Member Function Documentation

7.16.3.1 init()

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

Implements ISensor.

Definition at line 7 of file HMC5883L_WRAPPER.cpp.

7.16.3.2 read_data()

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

Implements ISensor.

Definition at line 12 of file HMC5883L_WRAPPER.cpp.

7.16.3.3 is_initialized()

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

Implements ISensor.

Definition at line 35 of file HMC5883L_WRAPPER.cpp.

7.16.3.4 get_type()

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

Implements ISensor.

Definition at line 39 of file HMC5883L_WRAPPER.cpp.

7.16.3.5 configure()

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

Implements ISensor.

Definition at line 43 of file HMC5883L_WRAPPER.cpp.

7.16.4 Member Data Documentation

7.16.4.1 sensor_

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

Definition at line 17 of file HMC5883L_WRAPPER.h.

7.16.4.2 initialized_

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

Definition at line 18 of file HMC5883L_WRAPPER.h.

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

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

7.17 INA3221 Class Reference

INA3221 Triple-Channel Power Monitor driver class.

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

Collaboration diagram for INA3221:

Classes

- struct conf_reg_t

  Configuration register bit fields.
- struct masken_reg_t

  Mask/Enable register bit fields.

Public Member Functions

- INA3221 (ina3221_addr_t addr, i2c_inst_t *i2c)

  Constructor for INA3221 class.
- bool begin ()

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

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

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

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

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

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

  Enable shunt voltage measurements.
- void set_shunt_measurement_disable ()

  Disable shunt voltage measurements.
- void set_bus_measurement_enable ()

  Enable bus voltage measurements.
- void set_bus_measurement_disable ()

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

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

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

  Set shunt voltage conversion time.
- uint16_t get_manufacturer_id ()

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

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

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

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

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

  Set warning alert voltage threshold for a channel.

- void set_crit_alert_limit (ina3221_ch_t channel, float voltage_v)

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

  Set power valid voltage range.
- void enable_alerts ()

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

  Get warning alert status for a channel.
- bool get_crit_alert (ina3221_ch_t channel)

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

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

  Set alert latch mode.

Private Member Functions

- void _read (ina3221_reg_t reg, uint16_t *val)

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

  Write a 16-bit value to a register.

Private Attributes

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

7.17.1 Detailed Description

INA3221 Triple-Channel Power Monitor driver class.

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

Definition at line 96 of file INA3221.h.

7.17.2 Member Function Documentation

7.17.2.1 _read()

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

Read a 16-bit register from the device.

Parameters

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

7.18.1 Detailed Description

Definition at line 33 of file ISensor.h.

7.18.2 Constructor & Destructor Documentation

7.18.2.1 ~ISensor()

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

7.18.3 Member Function Documentation

7.18.3.1 init()

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

Implemented in BH1750Wrapper, BME280Wrapper, HMC5883LWrapper, and MPU6050Wrapper.

7.18.3.2 read_data()

```
virtual float ISensor::read_data (
            SensorDataTypeIdentifier type) [pure virtual]
```

Implemented in BH1750Wrapper, BME280Wrapper, HMC5883LWrapper, and MPU6050Wrapper.

7.18.3.3 is_initialized()

```
virtual bool ISensor::is_initialized () const [pure virtual]
```

Implemented in BH1750Wrapper, BME280Wrapper, HMC5883LWrapper, and MPU6050Wrapper.

7.18.3.4 get_type()

```
virtual SensorType ISensor::get_type () const [pure virtual]
```

Implemented in BH1750Wrapper, BME280Wrapper, HMC5883LWrapper, and MPU6050Wrapper.

7.18.3.5 configure()

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

Implemented in BH1750Wrapper, BME280Wrapper, HMC5883LWrapper, and MPU6050Wrapper.

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

- lib/sensors/ISensor.h

7.19 LoRaClass Class Reference

```
#include <LoRa-RP2040.h>
```

Inheritance diagram for LoRaClass:

Collaboration diagram for LoRaClass:

Public Member Functions

- LoRaClass ()
- int begin (long frequency)
- void end ()
- int beginPacket (int implicitHeader=false)
- int endPacket (bool async=false)
- int parse_packet (int size=0)
- int packetRssi ()
- float packetSnr ()
- long packetFrequencyError ()
- int rssi ()
- virtual size_t write (uint8_t byte)
- virtual size_t write (const uint8_t *buffer, size_t size)
- virtual int available ()
- virtual int read ()
- virtual int peek ()
- virtual void flush ()
- void onCadDone (void(*callback)(bool))
- void onReceive (void(*callback)(int))
- void onTxDone (void(*callback)())
- void receive (int size=0)
- void channelActivityDetection (void)
- void idle ()
- void sleep ()
- void setTxPower (int level, int outputPin=PA_OUTPUT_PA_BOOST_PIN)
- void setFrequency (long frequency)
- void setSpreadingFactor (int sf)
- void setSignalBandwidth (long sbw)
- void setCodingRate4 (int denominator)
- void setPreambleLength (long length)
- void setSyncWord (int sw)
- void enableCrc ()
- void disableCrc ()
- void enableInvertIQ ()
- void disableInvertIQ ()
- void setOCP (uint8_t mA)
- void setGain (uint8_t gain)
- void crc ()
- void noCrc ()
- uint8_t random ()
- void set_pins (int ss=LORA_DEFAULT_SS_PIN, int reset=LORA_DEFAULT_RESET_PIN, int dio0=LORA_DEFAULT_DIO0_PIN
- void setSPI (spi_inst_t &spi)
- void setSPIFrequency (uint32_t frequency)
- void dumpRegisters ()

Public Member Functions inherited from Print

- Print ()
- int getWriteError ()
- void clearWriteError ()
- size_t write (const char *str)
- size_t write (const char *buffer, size_t size)
- virtual int availableForWrite ()
- size_t print (char)
- size_t print (const char *)
- size_t print (string c)
- size_t print (unsigned char, int=DEC)
- size_t print (int, int=DEC)
- size_t print (unsigned int, int=DEC)
- size_t print (long, int=DEC)
- size_t print (unsigned long, int=DEC)
- size_t print (long long, int=DEC)
- size_t print (unsigned long long, int=DEC)
- size_t print (double, int=2)
- size_t println (const char[ ])
- size_t println (char)
- size_t println (unsigned char, int=DEC)
- size_t println (int, int=DEC)
- size_t println (unsigned int, int=DEC)
- size_t println (long, int=DEC)
- size_t println (unsigned long, int=DEC)
- size_t println (long long, int=DEC)
- size_t println (unsigned long long, int=DEC)
- size_t println (double, int=2)
- size_t println (void)

Private Member Functions

- void explicitHeaderMode ()
- void implicitHeaderMode ()
- void handleDio0Rise ()
- bool isTransmitting ()
- int getSpreadingFactor ()
- long getSignalBandwidth ()
- void setLdoFlag ()
- uint8_t readRegister (uint8_t address)
- void writeRegister (uint8_t address, uint8_t value)
- uint8_t singleTransfer (uint8_t address, uint8_t value)

Static Private Member Functions

- static void onDio0Rise (uint, uint32_t)

Private Attributes

- spi_inst_t * _spi
- int _ss
- int _reset
- int _dio0
- long _frequency
- int _packetIndex
- int _implicitHeaderMode
- void(* _onReceive)(int)
- void(* _onCadDone)(bool)
- void(* _onTxDone)()

Additional Inherited Members

Protected Member Functions inherited from Print

- void setWriteError (int err=1)

7.19.1 Detailed Description

Definition at line 17 of file LoRa-RP2040.h.

7.19.2 Constructor & Destructor Documentation

7.19.2.1 LoRaClass()

```
LoRaClass::LoRaClass ()
```

Definition at line 69 of file LoRa-RP2040.cpp.

7.19.3 Member Function Documentation

7.19.3.1 begin()

```
int LoRaClass::begin (
            long frequency)
```

Definition at line 80 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.2 end()

```
void LoRaClass::end ()
```

Definition at line 149 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.3 beginPacket()

```
int LoRaClass::beginPacket (
            int implicitHeader = false)
```

Definition at line 158 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.4 endPacket()

```
int LoRaClass::endPacket (
            bool async = false)
```

Definition at line 180 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.5 parse_packet()

```
int LoRaClass::parse_packet (
            int size = 0)
```

Definition at line 215 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.6 packetRssi()

```
int LoRaClass::packetRssi ()
```

Definition at line 261 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.7 packetSnr()

```
float LoRaClass::packetSnr ()
```

Definition at line 266 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.8 packetFrequencyError()

```
long LoRaClass::packetFrequencyError ()
```

Definition at line 271 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.9 rssi()

```
int LoRaClass::rssi ()
```

Definition at line 290 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.10 write() [1/2]

```
size_t LoRaClass::write (
            uint8_t byte)  [virtual]
```

Implements Print.

Definition at line 295 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.11 write() [2/2]

```
size_t LoRaClass::write (
            const uint8_t * buffer,
            size_t size)  [virtual]
```

Reimplemented from Print.

Definition at line 300 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.12 available()

```
int LoRaClass::available ()  [virtual]
```

Definition at line 320 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.13 read()

```
int LoRaClass::read () [virtual]
```

Definition at line 325 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.14 peek()

```
int LoRaClass::peek () [virtual]
```

Definition at line 336 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.15 flush()

```
void LoRaClass::flush () [virtual]
```

Reimplemented from Print.

Definition at line 354 of file LoRa-RP2040.cpp.

7.19.3.16 onCadDone()

```
void LoRaClass::onCadDone (
            void(* callback )(bool))
```

Definition at line 369 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.17 onReceive()

```
void LoRaClass::onReceive (
            void(* callback )(int))
```

Definition at line 358 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.18 onTxDone()

```
void LoRaClass::onTxDone (
            void(* callback )())
```

Definition at line 381 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.19 receive()

```
void LoRaClass::receive (
            int size = 0)
```

Definition at line 392 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.20 channelActivityDetection()

```
void LoRaClass::channelActivityDetection (
            void )
```

Definition at line 408 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.21 idle()

```
void LoRaClass::idle ()
```

Definition at line 414 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.22 sleep()

```
void LoRaClass::sleep ()
```

Definition at line 419 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.23 setTxPower()

```
void LoRaClass::setTxPower (
            int level,
            int outputPin = PA_OUTPUT_PA_BOOST_PIN)
```

Definition at line 424 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.24 setFrequency()

```
void LoRaClass::setFrequency (
            long frequency)
```

Definition at line 461 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.25 setSpreadingFactor()

```
void LoRaClass::setSpreadingFactor (
            int sf)
```

Definition at line 477 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.26 setSignalBandwidth()

```
void LoRaClass::setSignalBandwidth (
            long sbw)
```

Definition at line 517 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.27 setCodingRate4()

```
void LoRaClass::setCodingRate4 (
            int denominator)
```

Definition at line 560 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.28 setPreambleLength()

```
void LoRaClass::setPreambleLength (
            long length)
```

Definition at line 573 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.29 setSyncWord()

```
void LoRaClass::setSyncWord (
            int sw)
```

Definition at line 579 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.30 enableCrc()

```
void LoRaClass::enableCrc ()
```

Definition at line 584 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.31 disableCrc()

```
void LoRaClass::disableCrc ()
```

Definition at line 589 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.32 enableInvertIQ()

```
void LoRaClass::enableInvertIQ ()
```

Definition at line 594 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.33 disableInvertIQ()

```
void LoRaClass::disableInvertIQ ()
```

Definition at line 600 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.34 setOCP()

```
void LoRaClass::setOCP (
            uint8_t mA)
```

Definition at line 606 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.35 setGain()

```
void LoRaClass::setGain (
            uint8_t gain)
```

Definition at line 619 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.36 crc()

```
void LoRaClass::crc () [inline]
```

Definition at line 73 of file LoRa-RP2040.h.

Here is the call graph for this function:

7.19.3.37 noCrc()

```
void LoRaClass::noCrc () [inline]
```

Definition at line 74 of file LoRa-RP2040.h.

Here is the call graph for this function:

7.19.3.38 random()

```
uint8_t LoRaClass::random ()
```

Definition at line 645 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.39 set_pins()

```
void LoRaClass::set_pins (
            int ss = LORA_DEFAULT_SS_PIN,
            int reset = LORA_DEFAULT_RESET_PIN,
            int dio0 = LORA_DEFAULT_DIO0_PIN)
```

Definition at line 650 of file LoRa-RP2040.cpp.

7.19.3.40 setSPI()

```
void LoRaClass::setSPI (
            spi_inst_t & spi)
```

Definition at line 657 of file LoRa-RP2040.cpp.

7.19.3.41 setSPIFrequency()

```
void LoRaClass::setSPIFrequency (
            uint32_t frequency)
```

Definition at line 662 of file LoRa-RP2040.cpp.

7.19.3.42 dumpRegisters()

```
void LoRaClass::dumpRegisters ()
```

Definition at line 667 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.43 explicitHeaderMode()

```
void LoRaClass::explicitHeaderMode () [private]
```

Definition at line 674 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.44 implicitHeaderMode()

```
void LoRaClass::implicitHeaderMode () [private]
```

Definition at line 681 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.45 handleDio0Rise()

```
void LoRaClass::handleDio0Rise () [private]
```

Definition at line 688 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

7.19.3.46 isTransmitting()

```
bool LoRaClass::isTransmitting () [private]
```

Definition at line 201 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.47 getSpreadingFactor()

```
int LoRaClass::getSpreadingFactor () [private]
```

Definition at line 472 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.48 getSignalBandwidth()

```
long LoRaClass::getSignalBandwidth () [private]
```

Definition at line 497 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.49 setLdoFlag()

```
void LoRaClass::setLdoFlag () [private]
```

Definition at line 547 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.50 readRegister()

```
uint8_t LoRaClass::readRegister (
            uint8_t address) [private]
```

Definition at line 723 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.51 writeRegister()

```
void LoRaClass::writeRegister (
            uint8_t address,
            uint8_t value)  [private]
```

Definition at line 728 of file LoRa-RP2040.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.19.3.52 singleTransfer()

```
uint8_t LoRaClass::singleTransfer (
            uint8_t address,
            uint8_t value)  [private]
```

Definition at line 733 of file LoRa-RP2040.cpp.

Here is the caller graph for this function:

7.19.3.53 onDio0Rise()

```
void LoRaClass::onDio0Rise (
            uint gpio,
            uint32_t events)  [static], [private]
```

Definition at line 747 of file LoRa-RP2040.cpp.

Here is the caller graph for this function:

7.19.4 Member Data Documentation

7.19.4.1 _spi

```
spi_inst_t* LoRaClass::_spi  [private]
```

Definition at line 104 of file LoRa-RP2040.h.

7.19.4.2 _ss

```
int LoRaClass::_ss  [private]
```

Definition at line 105 of file LoRa-RP2040.h.

7.19.4.3 _reset

```
int LoRaClass::_reset  [private]
```

Definition at line 106 of file LoRa-RP2040.h.

7.19.4.4 _dio0

```
int LoRaClass::_dio0  [private]
```

Definition at line 107 of file LoRa-RP2040.h.

7.19.4.5 _frequency

```
long LoRaClass::_frequency  [private]
```

Definition at line 108 of file LoRa-RP2040.h.

7.19.4.6 _packetIndex

```
int LoRaClass::_packetIndex [private]
```

Definition at line 109 of file LoRa-RP2040.h.

7.19.4.7 _implicitHeaderMode

```
int LoRaClass::_implicitHeaderMode [private]
```

Definition at line 110 of file LoRa-RP2040.h.

7.19.4.8 _onReceive

```
void(* LoRaClass::_onReceive) (int) [private]
```

Definition at line 111 of file LoRa-RP2040.h.

7.19.4.9 _onCadDone

```
void(* LoRaClass::_onCadDone) (bool) [private]
```

Definition at line 112 of file LoRa-RP2040.h.

7.19.4.10 _onTxDone

```
void(* LoRaClass::_onTxDone) () [private]
```

Definition at line 113 of file LoRa-RP2040.h.

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

- lib/comms/LoRa/LoRa-RP2040.h
- lib/comms/LoRa/LoRa-RP2040.cpp

7.20 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

7.20.1 Detailed Description

Mask/Enable register bit fields.

Definition at line 117 of file INA3221.h.

7.20.2 Member Data Documentation

7.20.2.1 conv_ready

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

Definition at line 118 of file INA3221.h.

7.20.2.2 timing_ctrl_alert

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

Definition at line 119 of file INA3221.h.

7.20.2.3 pwr_valid_alert

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

Definition at line 120 of file INA3221.h.

7.20.2.4 warn_alert_ch3

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

Definition at line 121 of file INA3221.h.

7.20.2.5 warn_alert_ch2

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

Definition at line 122 of file INA3221.h.

7.20.2.6 warn_alert_ch1

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

Definition at line 123 of file INA3221.h.

7.20.2.7 shunt_sum_alert

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

Definition at line 124 of file INA3221.h.

7.20.2.8 crit_alert_ch3

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

Definition at line 125 of file INA3221.h.

7.20.2.9 crit_alert_ch2

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

Definition at line 126 of file INA3221.h.

7.20.2.10 crit_alert_ch1

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

Definition at line 127 of file INA3221.h.

7.20.2.11 crit_alert_latch_en

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

Definition at line 128 of file INA3221.h.

7.20.2.12 warn_alert_latch_en

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

Definition at line 129 of file INA3221.h.

7.20.2.13 shunt_sum_en_ch3

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

Definition at line 130 of file INA3221.h.

7.20.2.14 shunt_sum_en_ch2

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

Definition at line 131 of file INA3221.h.

7.20.2.15 shunt_sum_en_ch1

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

Definition at line 132 of file INA3221.h.

7.20.2.16 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

7.21 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

Private Attributes

- MPU6050 sensor_
- bool initialized_ = false

7.21.1 Detailed Description

Definition at line 9 of file MPU6050_WRAPPER.h.

7.21.2 Constructor & Destructor Documentation

7.21.2.1 MPU6050Wrapper()

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

7.21.3 Member Function Documentation

7.21.3.1 init()

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

Implements ISensor.

7.21.3.2 read_data()

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

Implements ISensor.

7.21.3.3 is_initialized()

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

Implements ISensor.

7.21.3.4 get_type()

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

Implements ISensor.

7.21.3.5 configure()

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

Implements ISensor.

7.21.4 Member Data Documentation

7.21.4.1 sensor_

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

Definition at line 11 of file MPU6050_WRAPPER.h.

7.21.4.2 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

7.22 NMEAData Class Reference

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

Public Member Functions

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

Private Attributes

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

7.22.1 Detailed Description

Definition at line 9 of file NMEA_data.h.

7.22.2 Constructor & Destructor Documentation

7.22.2.1 NMEAData()

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

Definition at line 5 of file NMEA_data.cpp.

7.22.3 Member Function Documentation

7.22.3.1 update_rmc_tokens()

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

Definition at line 10 of file NMEA_data.cpp.

7.22.3.2 update_gga_tokens()

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

Definition at line 16 of file NMEA_data.cpp.

7.22.3.3 get_rmc_tokens()

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

Definition at line 22 of file NMEA_data.cpp.

7.22.3.4 get_gga_tokens()

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

Definition at line 29 of file NMEA_data.cpp.

7.22.4 Member Data Documentation

7.22.4.1 rmc_tokens_

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

Definition at line 19 of file NMEA_data.h.

7.22.4.2 gga_tokens_

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

Definition at line 20 of file NMEA_data.h.

7.22.4.3 rmc_mutex_

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

Definition at line 21 of file NMEA_data.h.

7.22.4.4 gga_mutex_

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

Definition at line 22 of file NMEA_data.h.

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

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

7.23 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 = 4.7f
- 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

7.23.1 Detailed Description

Definition at line 11 of file PowerManager.h.

7.23.2 Constructor & Destructor Documentation

7.23.2.1 PowerManager()

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

Definition at line 5 of file PowerManager.cpp.

7.23.3 Member Function Documentation

7.23.3.1 initialize()

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

Definition at line 10 of file PowerManager.cpp.

7.23.3.2 read_device_ids()

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

Definition at line 27 of file PowerManager.cpp.

7.23.3.3 get_current_charge_solar()

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

Definition at line 68 of file PowerManager.cpp.

Here is the caller graph for this function:

7.23.3.4 get_current_charge_usb()

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

Definition at line 52 of file PowerManager.cpp.

Here is the caller graph for this function:

7.23.3.5 get_current_charge_total()

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

Definition at line 76 of file PowerManager.cpp.

7.23.3.6 get_current_draw()

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

Definition at line 60 of file PowerManager.cpp.

7.23.3.7 get_voltage_battery()

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

Definition at line 36 of file PowerManager.cpp.

7.23.3.8 get_voltage_5v()

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

Definition at line 44 of file PowerManager.cpp.

Here is the caller graph for this function:

7.23.3.9 configure()

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

Definition at line 84 of file PowerManager.cpp.

7.23.3.10 is_charging_solar()

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

Definition at line 113 of file PowerManager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.23.3.11 is_charging_usb()

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

Definition at line 121 of file PowerManager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.23.3.12 check_power_alerts()

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

Definition at line 129 of file PowerManager.cpp.

Here is the call graph for this function:

7.23.4 Member Data Documentation

7.23.4.1 SOLAR_CURRENT_THRESHOLD

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

Definition at line 28 of file PowerManager.h.

7.23.4.2 USB_CURRENT_THRESHOLD

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

Definition at line 29 of file PowerManager.h.

7.23.4.3 VOLTAGE_LOW_THRESHOLD

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

Definition at line 30 of file PowerManager.h.

7.23.4.4 VOLTAGE_OVERCHARGE_THRESHOLD

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

Definition at line 31 of file PowerManager.h.

7.23.4.5 FALL_RATE_THRESHOLD

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

Definition at line 32 of file PowerManager.h.

7.23.4.6 FALLING_TREND_REQUIRED

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

Definition at line 33 of file PowerManager.h.

7.23.4.7 ina3221_

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

Definition at line 36 of file PowerManager.h.

7.23.4.8 initialized_

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

Definition at line 37 of file PowerManager.h.

7.23.4.9 powerman_mutex_

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

Definition at line 38 of file PowerManager.h.

7.23.4.10 charging_solar_active_

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

Definition at line 39 of file PowerManager.h.

7.23.4.11 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

7.24 Print Class Reference

```
#include <Print.h>
```

Inheritance diagram for Print:

Public Member Functions

- Print ()
- int getWriteError ()
- void clearWriteError ()
- virtual size_t write (uint8_t)=0
- size_t write (const char *str)
- virtual size_t write (const uint8_t *buffer, size_t size)
- size_t write (const char *buffer, size_t size)
- virtual int availableForWrite ()
- size_t print (char)
- size_t print (const char *)
- size_t print (string c)

- size_t print (unsigned char, int=DEC)
- size_t print (int, int=DEC)
- size_t print (unsigned int, int=DEC)
- size_t print (long, int=DEC)
- size_t print (unsigned long, int=DEC)
- size_t print (long long, int=DEC)
- size_t print (unsigned long long, int=DEC)
- size_t print (double, int=2)
- size_t println (const char[ ])
- size_t println (char)
- size_t println (unsigned char, int=DEC)
- size_t println (int, int=DEC)
- size_t println (unsigned int, int=DEC)
- size_t println (long, int=DEC)
- size_t println (unsigned long, int=DEC)
- size_t println (long long, int=DEC)
- size_t println (unsigned long long, int=DEC)
- size_t println (double, int=2)
- size_t println (void)
- virtual void flush ()

Protected Member Functions

- void setWriteError (int err=1)

Private Member Functions

- size_t printNumber (unsigned long, uint8_t)
- size_t printULLNumber (unsigned long long, uint8_t)
- size_t printFloat (double, int)

Private Attributes

- int write_error

7.24.1 Detailed Description

Definition at line 34 of file Print.h.

7.24.2 Constructor & Destructor Documentation

7.24.2.1 Print()

```
Print::Print () [inline]
```

Definition at line 44 of file Print.h.

7.24.3 Member Function Documentation

7.24.3.1 printNumber()

```
size_t Print::printNumber (
            unsigned long n,
            uint8_t base)  [private]
```

Definition at line 203 of file Print.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.3.2 printULLNumber()

```
size_t Print::printULLNumber (
            unsigned long long n64,
            uint8_t base)  [private]
```

Definition at line 246 of file Print.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.3.3 printFloat()

```
size_t Print::printFloat (
            double number,
            int digits)  [private]
```

Definition at line 298 of file Print.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.3.4 setWriteError()

```
void Print::setWriteError (
            int err = 1)  [inline], [protected]
```

Definition at line 42 of file Print.h.

Here is the caller graph for this function:

7.24.3.5 getWriteError()

```
int Print::getWriteError ()  [inline]
```

Definition at line 46 of file Print.h.

7.24.3.6 clearWriteError()

```
void Print::clearWriteError ()  [inline]
```

Definition at line 47 of file Print.h.

Here is the call graph for this function:

7.24.3.7 write() [1/4]

```
virtual size_t Print::write (
            uint8_t )  [pure virtual]
```

Implemented in LoRaClass.

Here is the caller graph for this function:

7.24.3.8 write() [2/4]

```
size_t Print::write (
            const char * str)  [inline]
```

Definition at line 50 of file Print.h.

Here is the call graph for this function:

7.24.3.9 write() [3/4]

```
size_t Print::write (
            const uint8_t * buffer,
            size_t size)  [virtual]
```

Reimplemented in LoRaClass.

Definition at line 31 of file Print.cpp.

Here is the call graph for this function:

7.24.3.10 write() [4/4]

```
size_t Print::write (
            const char * buffer,
            size_t size)  [inline]
```

Definition at line 55 of file Print.h.

Here is the call graph for this function:

7.24.3.11 availableForWrite()

```
virtual int Print::availableForWrite () [inline], [virtual]
```

Definition at line 61 of file Print.h.

7.24.3.12 print() [1/11]

```
size_t Print::print (
            char c)
```

Definition at line 51 of file Print.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.3.13 print() [2/11]

```
size_t Print::print (
            const char * c)
```

Definition at line 56 of file Print.cpp.

Here is the call graph for this function:

7.24.3.14 print() [3/11]

```
size_t Print::print (
            string c)
```

Definition at line 41 of file Print.cpp.

Here is the call graph for this function:

7.24.3.15 print() [4/11]

```
size_t Print::print (
            unsigned char b,
            int base = DEC)
```

Definition at line 61 of file Print.cpp.

Here is the call graph for this function:

7.24.3.16 print() [5/11]

```
size_t Print::print (
            int n,
            int base = DEC)
```

Definition at line 66 of file Print.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.3.17 print() [6/11]

```
size_t Print::print (
            unsigned int n,
            int base = DEC)
```

Definition at line 71 of file Print.cpp.

Here is the call graph for this function:

7.24.3.18 print() [7/11]

```
size_t Print::print (
            long n,
            int base = DEC)
```

Definition at line 76 of file Print.cpp.

Here is the call graph for this function:

7.24.3.19 print() [8/11]

```
size_t Print::print (
            unsigned long n,
            int base = DEC)
```

Definition at line 92 of file Print.cpp.

Here is the call graph for this function:

7.24.3.20 print() [9/11]

```
size_t Print::print (
            long long n,
            int base = DEC)
```

Definition at line 98 of file Print.cpp.

Here is the call graph for this function:

7.24.3.21 print() [10/11]

```
size_t Print::print (
            unsigned long long n,
            int base = DEC)
```

Definition at line 114 of file Print.cpp.

Here is the call graph for this function:

7.24.3.22 print() [11/11]

```
size_t Print::print (
            double n,
            int digits = 2)
```

Definition at line 120 of file Print.cpp.

Here is the call graph for this function:

7.24.3.23 println() [1/11]

```
size_t Print::println (
            const char c[])
```

Definition at line 130 of file Print.cpp.

Here is the call graph for this function:

7.24.3.24 println() [2/11]

```
size_t Print::println (
            char c)
```

Definition at line 137 of file Print.cpp.

Here is the call graph for this function:

7.24.3.25 println() [3/11]

```
size_t Print::println (
            unsigned char b,
            int base = DEC)
```

Definition at line 144 of file Print.cpp.

Here is the call graph for this function:

7.24.3.26 println() [4/11]

```
size_t Print::println (
            int num,
            int base = DEC)
```

Definition at line 151 of file Print.cpp.

Here is the call graph for this function:

7.24.3.27 println() [5/11]

```
size_t Print::println (
            unsigned int num,
            int base = DEC)
```

Definition at line 158 of file Print.cpp.

Here is the call graph for this function:

7.24.3.28 println() [6/11]

```
size_t Print::println (
            long num,
            int base = DEC)
```

Definition at line 165 of file Print.cpp.

Here is the call graph for this function:

7.24.3.29 println() [7/11]

```
size_t Print::println (
            unsigned long num,
            int base = DEC)
```

Definition at line 172 of file Print.cpp.

Here is the call graph for this function:

7.24.3.30 println() [8/11]

```
size_t Print::println (
            long long num,
            int base = DEC)
```

Definition at line 179 of file Print.cpp.

Here is the call graph for this function:

7.24.3.31 println() [9/11]

```
size_t Print::println (
            unsigned long long num,
            int base = DEC)
```

Definition at line 186 of file Print.cpp.

Here is the call graph for this function:

7.24.3.32 println() [10/11]

```
size_t Print::println (
            double num,
            int digits = 2)
```

Definition at line 193 of file Print.cpp.

Here is the call graph for this function:

7.24.3.33 println() [11/11]

```
size_t Print::println (
            void )
```

Definition at line 125 of file Print.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

7.24.3.34 flush()

```
virtual void Print::flush () [inline], [virtual]
```

Reimplemented in LoRaClass.

Definition at line 88 of file Print.h.

7.24.4 Member Data Documentation

7.24.4.1 write_error

```
int Print::write_error [private]
```

Definition at line 37 of file Print.h.

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

- lib/comms/LoRa/Print.h
- lib/comms/LoRa/Print.cpp

7.25 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.

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

7.25.1 Detailed Description

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

Definition at line 43 of file ISensor.h.

7.25.2 Constructor & Destructor Documentation

7.25.2.1 SensorWrapper()

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

Default constructor for SensorWrapper.

Here is the caller graph for this function:

7.25.3 Member Function Documentation

7.25.3.1 get_instance()

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

Provides a global instance of SensorWrapper.

Returns

A reference to the single SensorWrapper instance.

Definition at line 23 of file ISensor.cpp.

Here is the call graph for this function:

7.25.3.2 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 39 of file ISensor.cpp.

7.25.3.3 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 63 of file ISensor.cpp.

7.25.3.4 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 78 of file ISensor.cpp.

7.25.4 Member Data Documentation

7.25.4.1 sensors

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

Definition at line 51 of file ISensor.h.

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

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

Chapter 8

File Documentation

8.1 build_number.h File Reference

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

Macros

- #define BUILD_NUMBER 352

8.1.1 Macro Definition Documentation

8.1.1.1 BUILD_NUMBER

```
#define BUILD_NUMBER 352
```

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 352
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 "ISensor.h"
#include "lib/sensors/BH1750/BH1750_WRAPPER.h"
#include "lib/sensors/BME280/BME280_WRAPPER.h"
#include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
#include "lib/sensors/MPU6050/MPU6050_WRAPPER.h"
#include "lib/clock/DS3231.h"
#include <iostream>
#include <iomanip>
#include <queue>
#include <chrono>
#include "protocol.h"
#include <atomic>
#include <map>
#include "pin_config.h"
#include "utils.h"
#include "communication.h"
#include "build_number.h"
#include "lib/location/gps_collector.h"
#include "lib/storage/storage.h"
#include "lib/storage/pico-vfs/include/filesystem/vfs.h"
```

Include dependency graph for includes.h:

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

8.4 includes.h

Go to the documentation of this file.

```
#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 "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

#endif
```

8.5 lib/clock/DS3231.cpp File Reference

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

Include dependency graph for DS3231.cpp:

8.6 DS3231.cpp

Go to the documentation of this file.

```
#include "DS3231.h"
#include "utils.h"
#include <stdio.h> // Include for printf
```

```
#include <mutex>   // Include for mutex

DS3231::DS3231(i2c_inst_t *i2c_instance) : i2c(i2c_instance), ds3231_addr(DS3231_DEVICE_ADRESS) {
    // Initialize mutex (assuming you have a mutex member variable)
    recursive_mutex_init(&clock_mutex_);
}

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

    // Enable oscillator
    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;
    }

    status = "Raw BCD values read from DS3231: " + std::to_string(raw_data[0]) + " " +
            std::to_string(raw_data[1]) + " " + std::to_string(raw_data[2]) + " " +
            std::to_string(raw_data[3]) + " " + std::to_string(raw_data[4]) + " " +
            std::to_string(raw_data[5]) + " " + std::to_string(raw_data[6]);
    uart_print(status, VerbosityLevel::DEBUG);

    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]);

    // Data validation
    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;
    }

    uart_print("Reading time from DS3231", VerbosityLevel::DEBUG);
    std::string timeStr = "Time: " + std::to_string(data->hours) + ":" + std::to_string(data->minutes) + ":" + std::to_string(data->seconds);
    uart_print(timeStr, VerbosityLevel::DEBUG);
    std::string dateStr = "Date: " + std::to_string(data->date) + "/" + std::to_string(data->month) + "/" + std::to_string(data->year);
    uart_print(dateStr, VerbosityLevel::DEBUG);

    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]; // Signed for negative temperatures
    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::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);
}

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; // Indicate error
    }

    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;
}
```

8.7 lib/clock/DS3231.h File Reference

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

Include dependency graph for DS3231.h:

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

Classes

- struct ds3231_data_t
- class DS3231

Macros

- #define DS3231_DEVICE_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

Enumerations

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

8.7.1 Macro Definition Documentation

8.7.1.1 DS3231_DEVICE_ADRESS

```
#define DS3231_DEVICE_ADRESS 0x68
```

Definition at line 11 of file DS3231.h.

8.7.1.2 DS3231_SECONDS_REG

```
#define DS3231_SECONDS_REG 0x00
```

Definition at line 13 of file DS3231.h.

8.7.1.3 DS3231_MINUTES_REG

```
#define DS3231_MINUTES_REG 0x01
```

Definition at line 14 of file DS3231.h.

8.7.1.4 DS3231_HOURS_REG

```
#define DS3231_HOURS_REG 0x02
```

Definition at line 15 of file DS3231.h.

8.7.1.5 DS3231_DAY_REG

```
#define DS3231_DAY_REG 0x03
```

Definition at line 16 of file DS3231.h.

8.7.1.6 DS3231_DATE_REG

```
#define DS3231_DATE_REG 0x04
```

Definition at line 17 of file DS3231.h.

8.7.1.7 DS3231_MONTH_REG

```
#define DS3231_MONTH_REG 0x05
```

Definition at line 18 of file DS3231.h.

8.7.1.8 DS3231_YEAR_REG

```
#define DS3231_YEAR_REG 0x06
```

Definition at line 19 of file DS3231.h.

8.7.1.9 DS3231_CONTROL_REG

```
#define DS3231_CONTROL_REG 0x0E
```

Definition at line 21 of file DS3231.h.

8.7.1.10 DS3231_CONTROL_STATUS_REG

```
#define DS3231_CONTROL_STATUS_REG 0x0F
```

Definition at line 22 of file DS3231.h.

8.7.1.11 DS3231_TEMPERATURE_MSB_REG

```
#define DS3231_TEMPERATURE_MSB_REG 0x11
```

Definition at line 24 of file DS3231.h.

8.7.1.12 DS3231_TEMPERATURE_LSB_REG

```
#define DS3231_TEMPERATURE_LSB_REG 0x12
```

Definition at line 25 of file DS3231.h.

8.7.2 Enumeration Type Documentation

8.7.2.1 days_of_week

```
enum days_of_week
```

Enumerator

MONDAY	
TUESDAY	
WEDNESDAY	
THURSDAY	
FRIDAY	
SATURDAY	
SUNDAY	

Definition at line 27 of file DS3231.h.

8.8 DS3231.h

Go to the documentation of this file.

```
#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"

#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();

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_; // Mutex for I2C access
};

#endif
```

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

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

```
#include "DS3231.h"
```

Include dependency graph for clock_commands.cpp:

Macros

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

Functions

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

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

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

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

  Handler for getting last clock sync time.

Variables

- DS3231 systemClock

8.9.1 Macro Definition Documentation

8.9.1.1 CLOCK_GROUP

```
#define CLOCK_GROUP 3
```

Definition at line 5 of file clock_commands.cpp.

8.9.1.2 TIME

```
#define TIME 0
```

Definition at line 6 of file clock_commands.cpp.

8.9.1.3 TIMEZONE_OFFSET

```
#define TIMEZONE_OFFSET 1
```

Definition at line 7 of file clock_commands.cpp.

8.9.1.4 CLOCK_SYNC_INTERVAL

```
#define CLOCK_SYNC_INTERVAL 2
```

Definition at line 8 of file clock_commands.cpp.

8.9.1.5 LAST_SYNC_TIME

```
#define LAST_SYNC_TIME 3
```

Definition at line 9 of file clock_commands.cpp.

8.10 clock_commands.cpp

Go to the documentation of this file.

```
00001 #include "communication.h"
00002 #include <time.h>
00003 #include "DS3231.h" // Include the DS3231 header
00004
00005 #define CLOCK_GROUP 3
00006 #define TIME 0
00007 #define TIMEZONE_OFFSET 1
00008 #define CLOCK_SYNC_INTERVAL 2
00009 #define LAST_SYNC_TIME 3
00010
00016
00017  extern DS3231 systemClock;
00018
00032  Frame handle_time(const std::string& param, OperationType operationType) {
00033     // Validate operation type and parameter
00034     if (operationType == OperationType::SET) {
00035         if (param.empty()) return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "PARAM
      REQUIRED");
00036         try {
00037             time_t newTime = std::stoll(param);
00038             if (newTime <= 0) return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "TIME MUST
      BE POSITIVE");
00039
00040             if (systemClock.set_unix_time(newTime) != 0) return frame_build(ExecutionResult::ERROR,
      CLOCK_GROUP, TIME, "FAILED TO SET TIME");
00041
00042             EventEmitter::emit(EventGroup::CLOCK, ClockEvent::CHANGED);
00043             return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, TIME,
      std::to_string(systemClock.get_unix_time()));
00044
00045         } catch (...) {
00046             return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "INVALID TIME FORMAT");
00047         }
00048     } else if (operationType == OperationType::GET) {
00049         if (!param.empty()) return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "PARAM
      UNNECESSARY");
00050
00051         uint32_t timeUnix = systemClock.get_unix_time();
00052         if (timeUnix == 0) return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "FAILED TO
      GET TIME");
00053
00054         return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, TIME, std::to_string(timeUnix));
00055     }
00056
00057     return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "INVALID OPERATION");
00058 }
```

```cpp


Frame handle_timezone_offset(const std::string& param, OperationType operationType) {
    if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
        return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID OPERATION");
    }

    if (operationType == OperationType::GET) {
        if (!param.empty()) {
            return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "PARAM UNNECESSARY");
        }

        std::string timezoneOffset = "60";
        return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, TIMEZONE_OFFSET, timezoneOffset);
    }

    if (operationType == OperationType::SET) {
        if (param.empty()) {
            return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "PARAM REQUIRED");
        }
        try {
            int16_t offset = std::stoi(param);
            if (offset < -720 || offset > 720) { // ±12 hours in minutes
                return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID OFFSET");
            }

            // set offset
            return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, TIMEZONE_OFFSET, param);
        } catch (...) {
            return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID PARAMETER");
        }
    }
    return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "UNKNOWN ERROR");
}


Frame handle_clock_sync_interval(const std::string& param, OperationType operationType) {
    if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
        return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "INVALID OPERATION");
    }

    if (operationType == OperationType::GET) {
        if (!param.empty()) {
            return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "PARAM UNNECESSARY");
        }

        std::string clockSyncInterval = "1440";
        return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, clockSyncInterval);
    }

    if (operationType == OperationType::SET) {
        if (param.empty()) {
            return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "PARAM REQUIRED");
        }
        try {
            uint32_t interval = std::stoul(param);

            //set sync interval

            return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, param);
        } catch (...) {
            return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "INVALID PARAMETER");
        }
    }
    return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "UNKNOWN ERROR");
}

Frame handle_get_last_sync_time(const std::string& param, OperationType operationType) {
    if (operationType != OperationType::GET || !param.empty()) {
        return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, LAST_SYNC_TIME, "INVALID REQUEST");
    }
    std::string lastSyncTime = "none";
    return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, LAST_SYNC_TIME, lastSyncTime);
} // 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<Frame(const std::string&, OperationType)>

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

  Map type for storing command handlers.

Functions

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

  Executes a command based on its key.

Variables

- CommandMap commandHandlers

  Global map of all command handlers.

8.12 commands.cpp

Go to the documentation of this file.

```
// commands/commands.cpp
#include "commands.h"
#include "communication.h"


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

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

CommandMap commandHandlers = {
    {((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>(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>(1)), handle_file_download},
// Group 6, Command 1
    {((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},
// 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
};


Frame execute_command(uint32_t commandKey, const std::string& param, OperationType operationType) {
    auto it = commandHandlers.find(commandKey);
    if (it != commandHandlers.end()) {
        CommandHandler handler = it->second;
        return handler(param, operationType);
    } else {
        return frame_build(ExecutionResult::ERROR, 0, 0, "INVALID COMMAND");
    }
} // 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

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

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

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

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

  Handler for getting last clock sync time.
- Frame handle_get_commands_list (const std::string &param, OperationType operationType)

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

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

  Handles setting or getting the UART verbosity level.
- Frame handle_enter_bootloader_mode (const std::string &param, OperationType operationType)

  Reboot system to USB firmware loader.
- Frame handle_gps_power_status (const std::string &param, OperationType operationType)

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

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

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

  Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.
- Frame handle_get_power_manager_ids (const std::string &param, OperationType operationType)

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

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

  Handler for getting 5V rail voltage.
- Frame handle_get_current_charge_usb (const std::string &param, OperationType operationType)

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

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

Handler for getting total charge current.

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

  Handler for getting system current draw.
- Frame handle_get_last_events (const std::string &param, OperationType operationType)

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

  Handler for getting total number of events in the log.
- Frame handle_list_files (const std::string &param, OperationType operationType)

  Handles the list files command.
- Frame handle_file_download (const std::string &param, OperationType operationType)

  Handles the file download command.
- Frame handle_mount (const std::string &param, OperationType operationType)

  Handles the SD card mount/unmount command.
- 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< Frame(const std::string &, OperationType)> > commandHandlers

  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
Frame handle_time(const std::string& param, OperationType operationType);
Frame handle_timezone_offset(const std::string& param, OperationType operationType);
Frame handle_clock_sync_interval(const std::string& param, OperationType operationType);
Frame handle_get_last_sync_time(const std::string& param, OperationType operationType);

// DIAG
Frame handle_get_commands_list(const std::string& param, OperationType operationType);
Frame handle_get_build_version(const std::string& param, OperationType operationType);
Frame handle_verbosity(const std::string& param, OperationType operationType);
Frame handle_enter_bootloader_mode(const std::string& param, OperationType operationType);

// GPS
Frame handle_gps_power_status(const std::string& param, OperationType operationType);
Frame handle_enable_gps_uart_passthrough(const std::string& param, OperationType operationType);
Frame handle_get_rmc_data(const std::string& param, OperationType operationType);
Frame handle_get_gga_data(const std::string& param, OperationType operationType);

// POWER
Frame handle_get_power_manager_ids(const std::string& param, OperationType operationType);
Frame handle_get_voltage_battery(const std::string& param, OperationType operationType);
Frame handle_get_voltage_5v(const std::string& param, OperationType operationType);
Frame handle_get_current_charge_usb(const std::string& param, OperationType operationType);
Frame handle_get_current_charge_solar(const std::string& param, OperationType operationType);
Frame handle_get_current_charge_total(const std::string& param, OperationType operationType);
Frame handle_get_current_draw(const std::string& param, OperationType operationType);

// EVENT
Frame handle_get_last_events(const std::string& param, OperationType operationType);
Frame handle_get_event_count(const std::string& param, OperationType operationType);


//STORAGE
```

```
Frame handle_list_files(const std::string& param, OperationType operationType);
Frame handle_file_download(const std::string& param, OperationType operationType);
Frame handle_mount(const std::string& param, OperationType operationType);

Frame execute_command(uint32_t commandKey, const std::string& param, OperationType operationType);
extern std::map<uint32_t, std::function<Frame(const std::string&, OperationType)>> commandHandlers;

#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

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

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

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

  Handles setting or getting the UART verbosity level.
- Frame handle_enter_bootloader_mode (const std::string &param, OperationType operationType)

  Reboot system to USB firmware loader.

8.16 diagnostic_commands.cpp

Go to the documentation of this file.

```
#include "communication.h"
#include "commands.h"
#include "pico/stdlib.h"
#include "pico/bootrom.h"


Frame handle_get_commands_list(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 1, 0, "PARAM UNNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 1, 0, "INVALID OPERATION");
    }

    std::stringstream ss;
    for (const auto& entry : commandHandlers) {
        uint32_t commandKey = entry.first;
        uint8_t group = (commandKey >> 8) & 0xFF;
        uint8_t command = commandKey & 0xFF;

        ss << "Group: " << static_cast<int>(group)
```

```
            « ", Command: " « static_cast<int>(command) « "\n";
    }

    std::string commandList = ss.str();
    uart_print(commandList, VerbosityLevel::INFO); // Print to UART

    return frame_build(ExecutionResult::SUCCESS, 1, 0, "Commands listed on UART");
}

Frame handle_get_build_version(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 1, 1, "PARAM UNECESSARY");
    }
    if (operationType == OperationType::GET) {
        return frame_build(ExecutionResult::SUCCESS, 1, 1, std::to_string(BUILD_NUMBER));
    }
    return frame_build(ExecutionResult::ERROR, 1, 1, "INVALID OPERATION");
}


Frame handle_verbosity(const std::string& param, OperationType operationType) {
    if (param.empty()) {
        uart_print("Current verbosity level: " + std::to_string(static_cast<int>(g_uart_verbosity)),
    VerbosityLevel::INFO);
        return frame_build(ExecutionResult::SUCCESS, 1, 8,
                           std::to_string(static_cast<int>(g_uart_verbosity)));
    }

    try {
        int level = std::stoi(param);
        if (level < 0 || level > 5) {
            return frame_build(ExecutionResult::ERROR, 1, 8, "INVALID LEVEL (0-5)");
        }
        g_uart_verbosity = static_cast<VerbosityLevel>(level);
        return frame_build(ExecutionResult::SUCCESS, 1, 8, "LEVEL SET");
    } catch (...) {
        return frame_build(ExecutionResult::ERROR, 1, 8, "INVALID FORMAT");
    }
}

Frame handle_enter_bootloader_mode(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 1, 9, "PARAM UNNECESSARY");
    }

    if (operationType != OperationType::SET) {
        return frame_build(ExecutionResult::ERROR, 1, 9, "INVALID OPERATION");
    }

    // Build the success frame *before* resetting
    Frame successFrame = frame_build(ExecutionResult::SUCCESS, 1, 9, "REBOOT BOOTSEL");

    // Send the success frame
    uart_print("Sending BOOTSEL confirmation...");
    send_frame(successFrame); // Assuming you have a sendFrame function

    // Delay to ensure the frame is sent
    sleep_ms(100);

    uart_print("Entering BOOTSEL mode...", VerbosityLevel::WARNING);
    reset_usb_boot(0, 0); // Trigger BOOTSEL mode

    // The code will never reach here because the Pico will reset
    return frame_build(ExecutionResult::SUCCESS, 1, 9, "Entering BOOTSEL mode");
}

```

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

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

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

  Handler for getting total number of events in the log.

8.18 event_commands.cpp

Go to the documentation of this file.

```
00001 #include "communication.h"
00002 #include "event_manager.h"
00003 #include <sstream>
00004
00005
00011
00029 Frame handle_get_last_events(const std::string& param, OperationType operationType) {
00030     if (operationType != OperationType::GET) {
00031         return frame_build(ExecutionResult::ERROR, 5, 1, "INVALID OPERATION");
00032     }
00033
00034     size_t count = 10; // Default number of events to return
00035     if (!param.empty()) {
00036         try {
00037             count = std::stoul(param);
00038             if (count == 0 || count > EVENT_BUFFER_SIZE) {
00039                 return frame_build(ExecutionResult::ERROR, 5, 1, "INVALID COUNT");
00040             }
00041         } catch (...) {
00042             return frame_build(ExecutionResult::ERROR, 5, 1, "INVALID PARAMETER");
00043         }
00044     }
00045
00046     std::stringstream ss;
00047     ss « std::hex « std::uppercase « std::setfill('0');
00048
00049     size_t available = eventManager.get_event_count();
00050     size_t toReturn = std::min(count, available);
00051
00052     // Start from the most recent event
00053     for (size_t i = 0; i < toReturn; i++) {
00054         const EventLog& event = eventManager.get_event(available - 1 - i);
00055         // Format: IIIITTTTTTTTGGEE
00056         // IIII: 16-bit ID (4 hex chars)
00057         // TTTTTTTT: 32-bit timestamp (8 hex chars)
00058         // GG: 8-bit group (2 hex chars)
00059         // EE: 8-bit event (2 hex chars)
00060         ss « std::setw(4) « event.id
00061            « std::setw(8) « event.timestamp
00062            « std::setw(2) « static_cast<int>(event.group)
00063            « std::setw(2) « static_cast<int>(event.event);
00064         if (i < toReturn - 1) ss « "-";
00065     }
00066
00067     return frame_build(ExecutionResult::SUCCESS, 5, 1, ss.str());
00068 }
00069
00070
00083 Frame handle_get_event_count(const std::string& param, OperationType operationType) {
00084     if (operationType != OperationType::GET || !param.empty()) {
00085         return frame_build(ExecutionResult::ERROR, 5, 2, "INVALID REQUEST");
00086     }
00087
00088     return frame_build(ExecutionResult::SUCCESS, 5, 2,
00089                        std::to_string(eventManager.get_event_count()));
00090 } // end of EventCommands group
```

8.19 lib/comms/commands/gps_commands.cpp File Reference

```
#include "communication.h"
#include "lib/location/gps_collector.h"
#include <sstream>
```

Include dependency graph for gps_commands.cpp:

Functions

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

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

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

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

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

8.20 gps_commands.cpp

Go to the documentation of this file.

```
#include "communication.h"
#include "lib/location/gps_collector.h"
#include <sstream> // Include for stringstream


Frame handle_gps_power_status(const std::string& param, OperationType operationType) {
    if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
        return frame_build(ExecutionResult::ERROR, 7, 1, "INVALID OPERATION");
    }

    if (operationType == OperationType::SET) {
        if (param.empty()) {
            return frame_build(ExecutionResult::ERROR, 7, 1, "PARAM REQUIRED");
        }

        try {
            int powerStatus = std::stoi(param);
            if (powerStatus != 0 && powerStatus != 1) {
                return frame_build(ExecutionResult::ERROR, 7, 1, "INVALID VALUE. USE 0 OR 1");
            }
            gpio_put(GPS_POWER_ENABLE_PIN, powerStatus);
            EventEmitter::emit(EventGroup::GPS, powerStatus ? GPSEvent::POWER_ON :
GPSEvent::POWER_OFF);
            return frame_build(ExecutionResult::SUCCESS, 7, 1, std::to_string(powerStatus));
        } catch (...) {
            return frame_build(ExecutionResult::ERROR, 7, 1, "INVALID PARAMETER FORMAT");
        }
    }

    // GET operation
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 7, 1, "PARAM UNNECESSARY");
    }

    bool powerStatus = gpio_get(GPS_POWER_ENABLE_PIN);
    return frame_build(ExecutionResult::SUCCESS, 7, 1, std::to_string(powerStatus));
```

```
}


Frame handle_enable_gps_uart_passthrough(const std::string& param, OperationType operationType) {
    // Validate operation type
    if (!(operationType == OperationType::SET)) {
        return frame_build(ExecutionResult::ERROR, 7, 2, "NOT ALLOWED");
    }

    // Parse and validate timeout parameter
    uint32_t timeoutMs;
    try {
        timeoutMs = param.empty() ? 60000u : std::stoul(param) * 1000;
    } catch (...) {
        return frame_build(ExecutionResult::ERROR, 7, 2, "INVALID TIMEOUT FORMAT");
    }

    // Setup UART parameters and exit sequence
    const std::string EXIT_SEQUENCE = "##EXIT##";
    std::string inputBuffer;
    bool exitRequested = false;
    uint32_t originalBaudRate = DEBUG_UART_BAUD_RATE;
    uint32_t gpsBaudRate = GPS_UART_BAUD_RATE;
    uint32_t startTime = to_ms_since_boot(get_absolute_time());

    // Log start of transparent mode
    EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_START);

    // Print startup message
    std::string message = "Entering GPS Serial Pass-Through Mode @" +
                          std::to_string(gpsBaudRate) + " for " +
                          std::to_string(timeoutMs/1000) + "s\r\n" +
                          "Send " + EXIT_SEQUENCE + " to exit";
    uart_print(message, VerbosityLevel::INFO);

    // Allow time for message to be sent before baudrate change
    sleep_ms(10);

    // Switch to GPS baudrate
    uart_set_baudrate(DEBUG_UART_PORT, gpsBaudRate);

    // Main transparent mode loop
    while (!exitRequested) {
        while (uart_is_readable(DEBUG_UART_PORT)) {
            char ch = uart_getc(DEBUG_UART_PORT);

            inputBuffer += ch;
            if (inputBuffer.length() > EXIT_SEQUENCE.length()) {
                inputBuffer = inputBuffer.substr(1);
            }

            if (inputBuffer == EXIT_SEQUENCE) {
                exitRequested = true;
                break;
            }

            if (inputBuffer != EXIT_SEQUENCE.substr(0, inputBuffer.length())) {
                uart_write_blocking(GPS_UART_PORT,
                    reinterpret_cast<const uint8_t*>(&ch), 1);
            }
        }

        while (uart_is_readable(GPS_UART_PORT)) {
            char gpsByte = uart_getc(GPS_UART_PORT);
            uart_write_blocking(DEBUG_UART_PORT,
                reinterpret_cast<const uint8_t*>(&gpsByte), 1);
        }

        if (to_ms_since_boot(get_absolute_time()) - startTime >= timeoutMs) {
            break;
        }
    }

    uart_set_baudrate(DEBUG_UART_PORT, originalBaudRate);

    sleep_ms(10);

    EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_END);

    std::string exitReason = exitRequested ? "USER_EXIT" : "TIMEOUT";
    std::string response = "GPS UART BRIDGE EXIT: " + exitReason;
    uart_print(response, VerbosityLevel::INFO);

    return frame_build(ExecutionResult::SUCCESS, 7, 2, response);
}


```

```
Frame handle_get_rmc_data(const std::string& param, OperationType operationType) {
    if (operationType != OperationType::GET) {
        return frame_build(ExecutionResult::ERROR, 7, 3, "INVALID OPERATION");
    }

    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 7, 3, "PARAM UNNECESSARY");
    }

    std::vector<std::string> tokens = nmea_data.get_rmc_tokens();
    if (tokens.empty()) {
        return frame_build(ExecutionResult::ERROR, 7, 3, "NO RMC DATA");
    }

    // Join tokens with commas to create the response
    std::stringstream ss;
    for (size_t i = 0; i < tokens.size(); ++i) {
        ss « tokens[i];
        if (i < tokens.size() - 1) {
            ss « ",";
        }
    }

    return frame_build(ExecutionResult::SUCCESS, 7, 3, ss.str());
}


Frame handle_get_gga_data(const std::string& param, OperationType operationType) {
    if (operationType != OperationType::GET) {
        return frame_build(ExecutionResult::ERROR, 7, 4, "INVALID OPERATION");
    }

    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 7, 4, "PARAM UNNECESSARY");
    }

    std::vector<std::string> tokens = nmea_data.get_gga_tokens();
    if (tokens.empty()) {
        return frame_build(ExecutionResult::ERROR, 7, 4, "NO GGA DATA");
    }

    // Join tokens with commas to create the response
    std::stringstream ss;
    for (size_t i = 0; i < tokens.size(); ++i) {
        ss « tokens[i];
        if (i < tokens.size() - 1) {
            ss « ",";
        }
    }

    return frame_build(ExecutionResult::SUCCESS, 7, 4, ss.str());
} // end of GPSCommands group
```

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

`#include "communication.h"`

Include dependency graph for power_commands.cpp:

Functions

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

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

  Handler for getting battery voltage.

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

  Handler for getting 5V rail voltage.
- Frame handle_get_current_charge_usb (const std::string &param, OperationType operationType)

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

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

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

  Handler for getting system current draw.

8.22 power_commands.cpp

Go to the documentation of this file.

```
#include "communication.h"



Frame handle_get_power_manager_ids(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 2, 0, "PARAM UNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 2, 0, "INVALID OPERATION");
    }

    extern PowerManager powerManager;
    std::string powerManagerIDS = powerManager.read_device_ids();
    return frame_build(ExecutionResult::SUCCESS, 2, 0, powerManagerIDS);
}


Frame handle_get_voltage_battery(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 2, 2, "PARAM UNNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 2, 2, "NOT ALLOWED");
    }

    extern PowerManager powerManager;
    float voltage = powerManager.get_voltage_battery();
    return frame_build(ExecutionResult::SUCCESS, 2, 2, std::to_string(voltage), ValueUnit::VOLT);
}


Frame handle_get_voltage_5v(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 2, 3, "PARAM UNNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 2, 3, "NOT ALLOWED");
    }

    extern PowerManager powerManager;
    float voltage = powerManager.get_voltage_5v();
    return frame_build(ExecutionResult::SUCCESS, 2, 3, std::to_string(voltage), ValueUnit::VOLT);
}


Frame handle_get_current_charge_usb(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 2, 4, "PARAM UNNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 2, 4, "NOT ALLOWED");
    }

    extern PowerManager powerManager;
    float chargeCurrent = powerManager.get_current_charge_usb();
```

```
    return frame_build(ExecutionResult::SUCCESS, 2, 4, std::to_string(chargeCurrent), ValueUnit::MILIAMP);
}


Frame handle_get_current_charge_solar(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 2, 5, "PARAM UNNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 2, 5, "NOT ALLOWED");
    }

    extern PowerManager powerManager;
    float chargeCurrent = powerManager.get_current_charge_solar();
    return frame_build(ExecutionResult::SUCCESS, 2, 5, std::to_string(chargeCurrent), ValueUnit::MILIAMP);
}


Frame handle_get_current_charge_total(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 2, 6, "PARAM UNNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 2, 6, "NOT ALLOWED");
    }

    extern PowerManager powerManager;
    float chargeCurrent = powerManager.get_current_charge_total();
    return frame_build(ExecutionResult::SUCCESS, 2, 6, std::to_string(chargeCurrent), ValueUnit::MILIAMP);
}


Frame handle_get_current_draw(const std::string& param, OperationType operationType) {
    if (!param.empty()) {
        return frame_build(ExecutionResult::ERROR, 2, 7, "PARAM UNNECESSARY");
    }

    if (!(operationType == OperationType::GET)) {
        return frame_build(ExecutionResult::ERROR, 2, 7, "NOT ALLOWED");
    }


    extern PowerManager powerManager;
    float currentDraw = powerManager.get_current_draw();
    return frame_build(ExecutionResult::SUCCESS, 2, 7, std::to_string(currentDraw), ValueUnit::MILIAMP);
} // end of PowerCommands group
```

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

```
#include "commands.h"
#include "communication.h"
#include "storage.h"
#include "filesystem/vfs.h"
#include "filesystem/littlefs.h"
#include <sys/stat.h>
#include <errno.h>
#include "storage_commands_utils.h"
#include "dirent.h"
```

Include dependency graph for storage_commands.cpp:

Macros

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

Functions

- Frame handle_file_download (const std::string &param, OperationType operationType)

  Handles the file download command.
- Frame handle_list_files (const std::string &param, OperationType operationType)

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

  Handles the SD card mount/unmount command.

8.23.1 Macro Definition Documentation

8.23.1.1 MAX_BLOCK_SIZE

```
#define MAX_BLOCK_SIZE 250
```

Definition at line 11 of file storage_commands.cpp.

8.23.1.2 STORAGE_GROUP

```
#define STORAGE_GROUP 6
```

Definition at line 12 of file storage_commands.cpp.

8.23.1.3 START_COMMAND

```
#define START_COMMAND 1
```

Definition at line 13 of file storage_commands.cpp.

8.23.1.4 DATA_COMMAND

```
#define DATA_COMMAND 2
```

Definition at line 14 of file storage_commands.cpp.

8.23.1.5 END_COMMAND

```
#define END_COMMAND 3
```

Definition at line 15 of file storage_commands.cpp.

8.23.1.6 LIST_FILES_COMMAND

```
#define LIST_FILES_COMMAND 0
```

Definition at line 16 of file storage_commands.cpp.

8.23.1.7 MOUNT_COMMAND

```
#define MOUNT_COMMAND 4
```

Definition at line 17 of file storage_commands.cpp.

8.24 storage_commands.cpp

Go to the documentation of this file.

```
00001 #include "commands.h"
00002 #include "communication.h"
00003 #include "storage.h"
00004 #include "filesystem/vfs.h"
00005 #include "filesystem/littlefs.h"
00006 #include <sys/stat.h>
00007 #include <errno.h>
00008 #include "storage_commands_utils.h"
00009 #include "dirent.h"
00010
00011 #define MAX_BLOCK_SIZE  250
00012 #define STORAGE_GROUP 6
00013 #define START_COMMAND 1
00014 #define DATA_COMMAND 2
00015 #define END_COMMAND 3
00016 #define LIST_FILES_COMMAND 0
00017 #define MOUNT_COMMAND 4
00023
00043 Frame handle_file_download(const std::string& param, OperationType operationType) {
00044     if (operationType != OperationType::GET) {
00045         return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, START_COMMAND, "Invalid operation
      type");
00046     }
00047
00048     const char* filename = param.c_str();
00049     FILE* file = fopen(filename, "rb");
00050
00051     if (!file) {
00052         return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, START_COMMAND, "File not found");
00053     }
00054
00055     // Get file size
00056     fseek(file, 0, SEEK_END);
00057     size_t fileSize = ftell(file);
00058     fseek(file, 0, SEEK_SET);
00059
00060     // Send file size to ground station
00061     Frame sizeFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, START_COMMAND,
      std::to_string(fileSize));
00062     send_frame(sizeFrame);
00063
00064     size_t block_size = MAX_BLOCK_SIZE;
00065     size_t block_count = (fileSize + block_size - 1) / block_size;
00066
00067     // Send block size and count
```

```
    Frame blockSizeFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, START_COMMAND, std::to_string(block_size));
    send_frame(blockSizeFrame);
    Frame blockCountFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, START_COMMAND, std::to_string(block_count));
    send_frame(blockCountFrame);

    uint8_t buffer[MAX_BLOCK_SIZE];
    size_t bytesRead;
    uint32_t totalChecksum = 0;
    size_t blockIndex = 0;

    while ((bytesRead = fread(buffer, 1, MAX_BLOCK_SIZE, file)) > 0) {
        // Send data block
        send_data_block(buffer, bytesRead);
        totalChecksum = calculate_checksum(buffer, bytesRead);

        // Wait for ACK
        if (!receive_ack()) {
            fclose(file);
            return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, DATA_COMMAND, "ACK timeout");
        }
        blockIndex++;
    }

    fclose(file);

    // Send end frame with checksum
    std::stringstream ss;
    ss « std::hex « totalChecksum;
    Frame endFrame = frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, END_COMMAND, ss.str());
    send_frame(endFrame);

    return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, END_COMMAND, "File download complete");
}


Frame handle_list_files(const std::string& param, OperationType operationType) {
    if (operationType != OperationType::GET) {
        return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, LIST_FILES_COMMAND, "Invalid operation type");
    }

    DIR *dir;
    struct dirent *ent;
    if ((dir = opendir("/")) != 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 fileSize = 0;

            if (file != NULL) {
                fseek(file, 0, SEEK_END);
                fileSize = ftell(file);
                fclose(file);
            }

            // Create and send frame with filename and size
            char fileInfo[512];
            snprintf(fileInfo, sizeof(fileInfo), "%s:%zu", filename, fileSize);
            Frame fileFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, LIST_FILES_COMMAND, fileInfo);
            uart_print(fileInfo, VerbosityLevel::INFO);
            send_frame(fileFrame);
        }
        closedir(dir);
        return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, LIST_FILES_COMMAND, "File listing complete");
    } else {
        return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, LIST_FILES_COMMAND, "Could not open directory");
    }
}


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

```
    if (operationType != OperationType::SET) {
        return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Invalid operation type");
    }

    if (param == "1") {
        if (fs_init()) {
            return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, MOUNT_COMMAND, "SD card mounted");
        } else {
            return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Mount failed");
        }
    } else if (param == "0") {
        if (fs_unmount("/")) {
            sd_card_mounted = false;
            return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, MOUNT_COMMAND, "SD card unmounted");
        } else {
            return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Unmount failed");
        }
        return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Unmount not implemented");
    } else {
        return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Invalid parameter");
    }
} // StorageCommands
```

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

```
#include "communication.h"
#include "storage_commands_utils.h"
```

Include dependency graph for storage_commands_utils.cpp:

Functions

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

8.25.1 Function Documentation

8.25.1.1 calculate_checksum()

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

Definition at line 5 of file storage_commands_utils.cpp.

Here is the caller graph for this function:

8.25.1.2 send_data_block()

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

Definition at line 14 of file storage_commands_utils.cpp.

Here is the caller graph for this function:

8.25.1.3 receive_ack()

```
bool receive_ack ()
```

Definition at line 21 of file storage_commands_utils.cpp.

Here is the caller graph for this function:

8.26 storage_commands_utils.cpp

Go to the documentation of this file.

```
#include "communication.h"
#include "storage_commands_utils.h"

// Helper function to calculate checksum (simple XOR)
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;
}


void send_data_block(const uint8_t* data, size_t length) {
    LoRa.beginPacket();
    LoRa.write(data, length);
    LoRa.endPacket();
}

// Receiving an ACK (simplified)
bool receive_ack() {
    // Implement logic to receive an ACK frame from the ground station
    // Return true if ACK received, false otherwise
    // This is a placeholder, replace with your actual ACK receiving logic
    return true; // Placeholder: Always return true for now
}
```

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

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

Functions

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

8.27.1 Function Documentation

8.27.1.1 calculate_checksum()

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

Definition at line 5 of file storage_commands_utils.cpp.

Here is the caller graph for this function:

8.27.1.2 send_data_block()

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

Definition at line 14 of file storage_commands_utils.cpp.

Here is the caller graph for this function:

8.27.1.3 receive_ack()

```
bool receive_ack ()
```

Definition at line 21 of file storage_commands_utils.cpp.

Here is the caller graph for this function:

8.28 storage_commands_utils.h

Go to the documentation of this file.

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

8.29 lib/comms/communication.cpp File Reference

```
#include "communication.h"
```

Include dependency graph for communication.cpp:

Functions

- bool initialize_radio ()

  Initializes the LoRa radio module.

Variables

- string outgoing
- uint8_t msgCount = 0
- long lastSendTime = 0
- long lastReceiveTime = 0
- long lastPrintTime = 0
- unsigned long interval = 0

8.29.1 Function Documentation

8.29.1.1 initialize_radio()

```
bool initialize_radio ()
```

Initializes the LoRa radio module.

Returns

True if initialization was successful, false otherwise.

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

Definition at line 17 of file communication.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.29.2 Variable Documentation

8.29.2.1 outgoing

```
string outgoing
```

Definition at line 3 of file communication.cpp.

8.29.2.2 msgCount

```
uint8_t msgCount = 0
```

Definition at line 4 of file communication.cpp.

8.29.2.3 lastSendTime

```
long lastSendTime = 0
```

Definition at line 5 of file communication.cpp.

8.29.2.4 lastReceiveTime

```
long lastReceiveTime = 0
```

Definition at line 6 of file communication.cpp.

8.29.2.5 lastPrintTime

```
long lastPrintTime = 0
```

Definition at line 7 of file communication.cpp.

8.29.2.6 interval

```
unsigned long interval = 0
```

Definition at line 8 of file communication.cpp.

8.30 communication.cpp

Go to the documentation of this file.

```
00001 #include "communication.h"
00002
00003 string outgoing;
00004 uint8_t msgCount = 0;
00005 long lastSendTime = 0;
00006 long lastReceiveTime = 0;
00007 long lastPrintTime = 0;
00008 unsigned long interval = 0;
00009
00010
00017 bool initialize_radio() {
00018     LoRa.set_pins(lora_cs_pin, lora_reset_pin, lora_irq_pin);
00019     long frequency = 433E6;
00020     bool initStatus = false;
00021     if (!LoRa.begin(frequency))
00022     {
00023         uart_print("LoRa init failed. Check your connections.", VerbosityLevel::WARNING);
00024         initStatus = false;
00025     } else {
00026         uart_print("LoRa initialized with frequency " + std::to_string(frequency),
      VerbosityLevel::INFO);
00027         initStatus = true;
00028     }
00029
00030     EventEmitter::emit(EventGroup::COMMS, initStatus ? CommsEvent::RADIO_INIT :
      CommsEvent::RADIO_ERROR);
00031
00032     return initStatus;
00033 }
00034
```

8.31 lib/comms/communication.h File Reference

```
#include <string>
#include <vector>
#include "protocol.h"
#include "event_manager.h"
```

Include dependency graph for communication.h:

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

Functions

- bool initialize_radio ()

  Initializes the LoRa radio module.
- void on_receive (int packetSize)

  Callback function for handling received LoRa packets.
- void handle_uart_input ()

  Handles UART input.
- void send_message (std::string outgoing)
- void send_frame (const Frame &frame)
- void send_frame_uart (const Frame &frame)
- void send_frame_lora (const Frame &frame)
- void split_and_send_message (const uint8_t *data, size_t length)

  Sends a large packet using LoRa.
- 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)

  Converts a string into a Frame instance.
- Frame frame_build (ExecutionResult result, uint8_t group, uint8_t command, const std::string &value, const ValueUnit unitType=ValueUnit::UNDEFINED)

  Builds a Frame instance.
- std::string determine_unit (uint8_t group, uint8_t command)

8.31.1 Function Documentation

8.31.1.1 initialize_radio()

```
bool initialize_radio ()
```

Initializes the LoRa radio module.

Returns

True if initialization was successful, false otherwise.

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

Definition at line 17 of file communication.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.2 on_receive()

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

Callback function for handling received LoRa packets.

Parameters

packetSize	The size of the received packet.

Reads the received LoRa packet, extracts metadata, validates the lora_address_remote and local addresses, extracts the frame data, and processes it. Prints raw hex values for debugging.

Definition at line 15 of file receive.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.3 handle_uart_input()

```
void handle_uart_input ()
```

Handles UART input.

Reads characters from the UART port, appends them to a buffer, and processes the buffer when a newline character is received.

Definition at line 76 of file receive.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.4 send_message()

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

8.31.1.5 send_frame()

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

Definition at line 48 of file send.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.6 send_frame_uart()

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

Definition at line 42 of file send.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.7 send_frame_lora()

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

Definition at line 37 of file send.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.8 split_and_send_message()

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

Sends a large packet using LoRa.

Parameters

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 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 100 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.10 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.

Definition at line 19 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.11 frame_decode()

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

Converts a string into a Frame instance.

Parameters

data	The Frame data as a string.

Returns

The Frame instance.

Exceptions

std::runtime_error	if the frame header is invalid.

Parses the input string, extracting the frame direction, operation type, group, command, value, and unit. If an error occurs during parsing, an error message is printed to the UART, an error frame is built and sent, and the exception is re-thrown.

Definition at line 44 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.12 frame_build()

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

Builds a Frame instance.

Parameters

result	The execution result.
group	The group ID.
command	The command ID.
value	The value.
unitType	The value unit type.

Returns

The Frame instance.

Constructs a Frame instance based on the provided parameters. The frame direction and operation type are set based on the execution result.

Definition at line 135 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.31.1.13 determine_unit()

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

8.32 communication.h

Go to the documentation of this file.

```
#ifndef COMMUNICATION_H
#define COMMUNICATION_H

#include <string>
#include <vector>
#include "protocol.h"
#include "event_manager.h"

bool initialize_radio();
void on_receive(int packetSize);
void handle_uart_input();
void send_message(std::string outgoing);
void send_frame(const Frame& frame);
void send_frame_uart(const Frame& frame);
void send_frame_lora(const Frame& frame);

void split_and_send_message(const uint8_t* data, size_t length);

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(ExecutionResult result, 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::string(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)

  Converts 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 (ExecutionResult result, uint8_t group, uint8_t command, const std::string &value, const ValueUnit unitType)

  Builds a Frame instance.

Variables

- std::map< uint32_t, CommandHandler > commandHandlers

  Global map of all command handlers.
- volatile uint16_t eventRegister

8.33.1 Detailed Description

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

Definition in file frame.cpp.

8.33.2 Typedef Documentation

8.33.2.1 CommandHandler

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

Definition at line 3 of file frame.cpp.

8.33.3 Function Documentation

8.33.3.1 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.

Definition at line 19 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.33.3.2 frame_decode()

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

Converts a string into a Frame instance.

Parameters

data	The Frame data as a string.

Returns

The Frame instance.

Exceptions

std::runtime_error	if the frame header is invalid.

Parses the input string, extracting the frame direction, operation type, group, command, value, and unit. If an error occurs during parsing, an error message is printed to the UART, an error frame is built and sent, and the exception is re-thrown.

Definition at line 44 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.33.3.3 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 100 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.33.3.4 frame_build()

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

Builds a Frame instance.

Parameters

result	The execution result.
group	The group ID.
command	The command ID.
value	The value.
unitType	The value unit type.

Returns

The Frame instance.

Constructs a Frame instance based on the provided parameters. The frame direction and operation type are set based on the execution result.

Definition at line 135 of file frame.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.33.4 Variable Documentation

8.33.4.1 eventRegister

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

8.34 frame.cpp

Go to the documentation of this file.

```
#include "communication.h"

using CommandHandler = std::function<std::string(const std::string&, OperationType)>;
extern std::map<uint32_t, CommandHandler> commandHandlers;
extern volatile uint16_t eventRegister;


std::string frame_encode(const Frame& frame) {
    std::stringstream ss;
    ss « static_cast<int>(frame.direction) « DELIMITER
       « operation_type_to_string(frame.operationType) « DELIMITER
       « static_cast<int>(frame.group) « DELIMITER
       « static_cast<int>(frame.command) « DELIMITER
       « frame.value;

    if (!frame.unit.empty()) {
        ss « DELIMITER « frame.unit;
    }

    return FRAME_BEGIN + DELIMITER + ss.str() + DELIMITER + FRAME_END;
}


Frame frame_decode(const std::string& data) {
    try {
        Frame frame;
        std::stringstream ss(data);
        std::string token;

        std::getline(ss, token, DELIMITER);
        if (token != FRAME_BEGIN)
            throw std::runtime_error("Invalid frame header");
        frame.header = token;

        std::string decodedFrameData;
        while (std::getline(ss, token, DELIMITER)) {
            if (token == FRAME_END) break;
            decodedFrameData += token + DELIMITER;
        }
        if (!decodedFrameData.empty()) {
            decodedFrameData.pop_back();
        }

        std::stringstream frameDataStream(decodedFrameData);

        std::getline(frameDataStream, token, DELIMITER);
        frame.direction = std::stoi(token);

        std::getline(frameDataStream, token, DELIMITER);
        frame.operationType = string_to_operation_type(token);

        std::getline(frameDataStream, token, DELIMITER);
        frame.group = std::stoi(token);

        std::getline(frameDataStream, token, DELIMITER);
        frame.command = std::stoi(token);

        std::getline(frameDataStream, token, DELIMITER);
        frame.value = token;

        std::getline(frameDataStream, token, DELIMITER);
        frame.unit = token;

        return frame;
    } catch (const std::exception& e) {
        uart_print("Frame error: " + std::string(e.what()), VerbosityLevel::ERROR);
        Frame errorFrame = frame_build(ExecutionResult::ERROR, 0, 0, e.what());
        send_frame(errorFrame);
        throw;
    }
}


void frame_process(const std::string& data, Interface interface) {
    try {
        Frame frame = frame_decode(data);
        uint32_t commandKey = (static_cast<uint32_t>(frame.group) « 8) |
static_cast<uint32_t>(frame.command);

        Frame responseFrame = execute_command(commandKey, frame.value, frame.operationType);

        // Send response through the same interface that received the command
```

```
        if (interface == Interface::UART) {
            send_frame_uart(responseFrame);
        } else if (interface == Interface::LORA) {
            send_frame_lora(responseFrame);
        }
    } catch (const std::exception& e) {
        Frame errorFrame = frame_build(ExecutionResult::ERROR, 0, 0, e.what());
        // Send error through the same interface
        if (interface == Interface::UART) {
            send_frame_uart(errorFrame);
        } else if (interface == Interface::LORA) {
            send_frame_lora(errorFrame);
        }
    }
}

Frame frame_build(ExecutionResult result, uint8_t group, uint8_t command,
                const std::string& value, const ValueUnit unitType) {
    Frame frame;
    frame.header = FRAME_BEGIN;
    frame.footer = FRAME_END;

    switch (result) {
        case ExecutionResult::SUCCESS:
            frame.direction = 1;
            frame.operationType = OperationType::ANS;
            frame.value = value;
            frame.unit = value_unit_type_to_string(unitType);
            break;

        case ExecutionResult::ERROR:
            frame.direction = 1;
            frame.operationType = OperationType::ERR;
            frame.value = value;
            frame.unit = value_unit_type_to_string(ValueUnit::UNDEFINED);
            break;

        case ExecutionResult::INFO:
            frame.direction = 1;
            frame.operationType = OperationType::INF;
            frame.value = value;
            frame.unit = value_unit_type_to_string(ValueUnit::UNDEFINED);
            break;
    }

    frame.group = group;
    frame.command = command;

    return frame;
}
```

8.35 lib/comms/LoRa/LoRa-RP2040.cpp File Reference

`#include "LoRa-RP2040.h"`

Include dependency graph for LoRa-RP2040.cpp:

Macros

- #define REG_FIFO 0x00

- #define REG_OP_MODE 0x01
- #define REG_FRF_MSB 0x06
- #define REG_FRF_MID 0x07
- #define REG_FRF_LSB 0x08
- #define REG_PA_CONFIG 0x09
- #define IRQ_CAD_DETECTED_MASK 0x01
- #define REG_OCP 0x0b
- #define REG_LNA 0x0c
- #define IRQ_CAD_DONE_MASK 0x04
- #define REG_FIFO_ADDR_PTR 0x0d
- #define REG_FIFO_TX_BASE_ADDR 0x0e
- #define REG_FIFO_RX_BASE_ADDR 0x0f
- #define REG_FIFO_RX_CURRENT_ADDR 0x10
- #define REG_IRQ_FLAGS 0x12
- #define REG_RX_NB_BYTES 0x13
- #define REG_PKT_SNR_VALUE 0x19
- #define REG_PKT_RSSI_VALUE 0x1a
- #define REG_RSSI_VALUE 0x1b
- #define REG_MODEM_CONFIG_1 0x1d
- #define REG_MODEM_CONFIG_2 0x1e
- #define REG_PREAMBLE_MSB 0x20
- #define REG_PREAMBLE_LSB 0x21
- #define REG_PAYLOAD_LENGTH 0x22
- #define REG_MODEM_CONFIG_3 0x26
- #define REG_FREQ_ERROR_MSB 0x28
- #define REG_FREQ_ERROR_MID 0x29
- #define REG_FREQ_ERROR_LSB 0x2a
- #define REG_RSSI_WIDEBAND 0x2c
- #define REG_DETECTION_OPTIMIZE 0x31
- #define REG_INVERTIQ 0x33
- #define REG_DETECTION_THRESHOLD 0x37
- #define REG_SYNC_WORD 0x39
- #define REG_INVERTIQ2 0x3b
- #define REG_DIO_MAPPING_1 0x40
- #define REG_VERSION 0x42
- #define REG_PA_DAC 0x4d
- #define MODE_CAD 0x07
- #define MODE_LONG_RANGE_MODE 0x80
- #define MODE_SLEEP 0x00
- #define MODE_STDBY 0x01
- #define MODE_TX 0x03
- #define MODE_RX_CONTINUOUS 0x05
- #define MODE_RX_SINGLE 0x06
- #define PA_BOOST 0x80
- #define IRQ_TX_DONE_MASK 0x08
- #define IRQ_PAYLOAD_CRC_ERROR_MASK 0x20
- #define IRQ_RX_DONE_MASK 0x40
- #define RF_MID_BAND_THRESHOLD 525E6
- #define RSSI_OFFSET_HF_PORT 157
- #define RSSI_OFFSET_LF_PORT 164
- #define MAX_PKT_LENGTH 255
- #define ISR_PREFIX

Variables

- LoRaClass LoRa

8.35.1 Macro Definition Documentation

8.35.1.1 REG_FIFO

```
#define REG_FIFO 0x00
```

Definition at line 3 of file LoRa-RP2040.cpp.

8.35.1.2 REG_OP_MODE

```
#define REG_OP_MODE 0x01
```

Definition at line 4 of file LoRa-RP2040.cpp.

8.35.1.3 REG_FRF_MSB

```
#define REG_FRF_MSB 0x06
```

Definition at line 5 of file LoRa-RP2040.cpp.

8.35.1.4 REG_FRF_MID

```
#define REG_FRF_MID 0x07
```

Definition at line 6 of file LoRa-RP2040.cpp.

8.35.1.5 REG_FRF_LSB

```
#define REG_FRF_LSB 0x08
```

Definition at line 7 of file LoRa-RP2040.cpp.

8.35.1.6 REG_PA_CONFIG

```
#define REG_PA_CONFIG 0x09
```

Definition at line 8 of file LoRa-RP2040.cpp.

8.35.1.7 IRQ_CAD_DETECTED_MASK

```
#define IRQ_CAD_DETECTED_MASK 0x01
```

Definition at line 9 of file LoRa-RP2040.cpp.

8.35.1.8 REG_OCP

```
#define REG_OCP 0x0b
```

Definition at line 10 of file LoRa-RP2040.cpp.

8.35.1.9 REG_LNA

```
#define REG_LNA 0x0c
```

Definition at line 11 of file LoRa-RP2040.cpp.

8.35.1.10 IRQ_CAD_DONE_MASK

```
#define IRQ_CAD_DONE_MASK 0x04
```

Definition at line 12 of file LoRa-RP2040.cpp.

8.35.1.11 REG_FIFO_ADDR_PTR

```
#define REG_FIFO_ADDR_PTR 0x0d
```

Definition at line 13 of file LoRa-RP2040.cpp.

8.35.1.12 REG_FIFO_TX_BASE_ADDR

```
#define REG_FIFO_TX_BASE_ADDR 0x0e
```

Definition at line 14 of file LoRa-RP2040.cpp.

8.35.1.13 REG_FIFO_RX_BASE_ADDR

```
#define REG_FIFO_RX_BASE_ADDR 0x0f
```

Definition at line 15 of file LoRa-RP2040.cpp.

8.35.1.14 REG_FIFO_RX_CURRENT_ADDR

```
#define REG_FIFO_RX_CURRENT_ADDR 0x10
```

Definition at line 16 of file LoRa-RP2040.cpp.

8.35.1.15 REG_IRQ_FLAGS

```
#define REG_IRQ_FLAGS 0x12
```

Definition at line 17 of file LoRa-RP2040.cpp.

8.35.1.16 REG_RX_NB_BYTES

```
#define REG_RX_NB_BYTES 0x13
```

Definition at line 18 of file LoRa-RP2040.cpp.

8.35.1.17 REG_PKT_SNR_VALUE

```
#define REG_PKT_SNR_VALUE 0x19
```

Definition at line 19 of file LoRa-RP2040.cpp.

8.35.1.18 REG_PKT_RSSI_VALUE

```
#define REG_PKT_RSSI_VALUE 0x1a
```

Definition at line 20 of file LoRa-RP2040.cpp.

8.35.1.19 REG_RSSI_VALUE

```
#define REG_RSSI_VALUE 0x1b
```

Definition at line 21 of file LoRa-RP2040.cpp.

8.35.1.20 REG_MODEM_CONFIG_1

```
#define REG_MODEM_CONFIG_1 0x1d
```

Definition at line 22 of file LoRa-RP2040.cpp.

8.35.1.21 REG_MODEM_CONFIG_2

```
#define REG_MODEM_CONFIG_2 0x1e
```

Definition at line 23 of file LoRa-RP2040.cpp.

8.35.1.22 REG_PREAMBLE_MSB

```
#define REG_PREAMBLE_MSB 0x20
```

Definition at line 24 of file LoRa-RP2040.cpp.

8.35.1.23 REG_PREAMBLE_LSB

```
#define REG_PREAMBLE_LSB 0x21
```

Definition at line 25 of file LoRa-RP2040.cpp.

8.35.1.24 REG_PAYLOAD_LENGTH

```
#define REG_PAYLOAD_LENGTH 0x22
```

Definition at line 26 of file LoRa-RP2040.cpp.

8.35.1.25 REG_MODEM_CONFIG_3

```
#define REG_MODEM_CONFIG_3 0x26
```

Definition at line 27 of file LoRa-RP2040.cpp.

8.35.1.26 REG_FREQ_ERROR_MSB

```
#define REG_FREQ_ERROR_MSB 0x28
```

Definition at line 28 of file LoRa-RP2040.cpp.

8.35.1.27 REG_FREQ_ERROR_MID

```
#define REG_FREQ_ERROR_MID 0x29
```

Definition at line 29 of file LoRa-RP2040.cpp.

8.35.1.28 REG_FREQ_ERROR_LSB

```
#define REG_FREQ_ERROR_LSB 0x2a
```

Definition at line 30 of file LoRa-RP2040.cpp.

8.35.1.29 REG_RSSI_WIDEBAND

```
#define REG_RSSI_WIDEBAND 0x2c
```

Definition at line 31 of file LoRa-RP2040.cpp.

8.35.1.30 REG_DETECTION_OPTIMIZE

```
#define REG_DETECTION_OPTIMIZE 0x31
```

Definition at line 32 of file LoRa-RP2040.cpp.

8.35.1.31 REG_INVERTIQ

```
#define REG_INVERTIQ 0x33
```

Definition at line 33 of file LoRa-RP2040.cpp.

8.35.1.32 REG_DETECTION_THRESHOLD

```
#define REG_DETECTION_THRESHOLD 0x37
```

Definition at line 34 of file LoRa-RP2040.cpp.

8.35.1.33 REG_SYNC_WORD

```
#define REG_SYNC_WORD 0x39
```

Definition at line 35 of file LoRa-RP2040.cpp.

8.35.1.34 REG_INVERTIQ2

```
#define REG_INVERTIQ2 0x3b
```

Definition at line 36 of file LoRa-RP2040.cpp.

8.35.1.35 REG_DIO_MAPPING_1

```
#define REG_DIO_MAPPING_1 0x40
```

Definition at line 37 of file LoRa-RP2040.cpp.

8.35.1.36 REG_VERSION

```
#define REG_VERSION 0x42
```

Definition at line 38 of file LoRa-RP2040.cpp.

8.35.1.37 REG_PA_DAC

```
#define REG_PA_DAC 0x4d
```

Definition at line 39 of file LoRa-RP2040.cpp.

8.35.1.38 MODE_CAD

```
#define MODE_CAD 0x07
```

Definition at line 40 of file LoRa-RP2040.cpp.

8.35.1.39 MODE_LONG_RANGE_MODE

```
#define MODE_LONG_RANGE_MODE 0x80
```

Definition at line 42 of file LoRa-RP2040.cpp.

8.35.1.40 MODE_SLEEP

```
#define MODE_SLEEP 0x00
```

Definition at line 43 of file LoRa-RP2040.cpp.

8.35.1.41 MODE_STDBY

```
#define MODE_STDBY 0x01
```

Definition at line 44 of file LoRa-RP2040.cpp.

8.35.1.42 MODE_TX

```
#define MODE_TX 0x03
```

Definition at line 45 of file LoRa-RP2040.cpp.

8.35.1.43 MODE_RX_CONTINUOUS

```
#define MODE_RX_CONTINUOUS 0x05
```

Definition at line 46 of file LoRa-RP2040.cpp.

8.35.1.44 MODE_RX_SINGLE

```
#define MODE_RX_SINGLE 0x06
```

Definition at line 47 of file LoRa-RP2040.cpp.

8.35.1.45 PA_BOOST

```
#define PA_BOOST 0x80
```

Definition at line 50 of file LoRa-RP2040.cpp.

8.35.1.46 IRQ_TX_DONE_MASK

```
#define IRQ_TX_DONE_MASK 0x08
```

Definition at line 53 of file LoRa-RP2040.cpp.

8.35.1.47 IRQ_PAYLOAD_CRC_ERROR_MASK

```
#define IRQ_PAYLOAD_CRC_ERROR_MASK 0x20
```

Definition at line 54 of file LoRa-RP2040.cpp.

8.35.1.48 IRQ_RX_DONE_MASK

```
#define IRQ_RX_DONE_MASK 0x40
```

Definition at line 55 of file LoRa-RP2040.cpp.

8.35.1.49 RF_MID_BAND_THRESHOLD

```
#define RF_MID_BAND_THRESHOLD 525E6
```

Definition at line 57 of file LoRa-RP2040.cpp.

8.35.1.50 RSSI_OFFSET_HF_PORT

```
#define RSSI_OFFSET_HF_PORT 157
```

Definition at line 58 of file LoRa-RP2040.cpp.

8.35.1.51 RSSI_OFFSET_LF_PORT

```
#define RSSI_OFFSET_LF_PORT 164
```

Definition at line 59 of file LoRa-RP2040.cpp.

8.35.1.52 MAX_PKT_LENGTH

```
#define MAX_PKT_LENGTH 255
```

Definition at line 61 of file LoRa-RP2040.cpp.

8.35.1.53 ISR_PREFIX

```
#define ISR_PREFIX
```

Definition at line 66 of file LoRa-RP2040.cpp.

8.35.2 Variable Documentation

8.35.2.1 LoRa

```
LoRaClass LoRa
```

Definition at line 753 of file LoRa-RP2040.cpp.

8.36 LoRa-RP2040.cpp

Go to the documentation of this file.

```
#include "LoRa-RP2040.h"
// registers
#define REG_FIFO                 0x00
#define REG_OP_MODE              0x01
#define REG_FRF_MSB              0x06
#define REG_FRF_MID              0x07
#define REG_FRF_LSB              0x08
#define REG_PA_CONFIG            0x09
#define IRQ_CAD_DETECTED_MASK    0x01
#define REG_OCP                  0x0b
#define REG_LNA                  0x0c
#define IRQ_CAD_DONE_MASK        0x04
#define REG_FIFO_ADDR_PTR        0x0d
#define REG_FIFO_TX_BASE_ADDR    0x0e
#define REG_FIFO_RX_BASE_ADDR    0x0f
#define REG_FIFO_RX_CURRENT_ADDR 0x10
#define REG_IRQ_FLAGS            0x12
#define REG_RX_NB_BYTES          0x13
#define REG_PKT_SNR_VALUE        0x19
#define REG_PKT_RSSI_VALUE       0x1a
#define REG_RSSI_VALUE           0x1b
#define REG_MODEM_CONFIG_1       0x1d
#define REG_MODEM_CONFIG_2       0x1e
#define REG_PREAMBLE_MSB         0x20
#define REG_PREAMBLE_LSB         0x21
#define REG_PAYLOAD_LENGTH       0x22
#define REG_MODEM_CONFIG_3       0x26
#define REG_FREQ_ERROR_MSB       0x28
#define REG_FREQ_ERROR_MID       0x29
#define REG_FREQ_ERROR_LSB       0x2a
#define REG_RSSI_WIDEBAND        0x2c
#define REG_DETECTION_OPTIMIZE   0x31
#define REG_INVERTIQ             0x33
#define REG_DETECTION_THRESHOLD  0x37
#define REG_SYNC_WORD            0x39
#define REG_INVERTIQ2            0x3b
#define REG_DIO_MAPPING_1        0x40
#define REG_VERSION              0x42
#define REG_PA_DAC               0x4d
#define MODE_CAD                 0x07
// modes
#define MODE_LONG_RANGE_MODE     0x80
#define MODE_SLEEP               0x00
#define MODE_STDBY               0x01
#define MODE_TX                  0x03
#define MODE_RX_CONTINUOUS       0x05
#define MODE_RX_SINGLE           0x06

// PA config
#define PA_BOOST                 0x80

// IRQ masks
#define IRQ_TX_DONE_MASK           0x08
#define IRQ_PAYLOAD_CRC_ERROR_MASK 0x20
#define IRQ_RX_DONE_MASK           0x40

#define RF_MID_BAND_THRESHOLD    525E6
#define RSSI_OFFSET_HF_PORT      157
#define RSSI_OFFSET_LF_PORT      164

#define MAX_PKT_LENGTH           255

#if (ESP8266 || ESP32)
#define ISR_PREFIX ICACHE_RAM_ATTR
#else
#define ISR_PREFIX
#endif

LoRaClass::LoRaClass() :
      _spi(SPI_PORT),
      _ss(LORA_DEFAULT_SS_PIN), _reset(LORA_DEFAULT_RESET_PIN), _dio0(LORA_DEFAULT_DIO0_PIN),
      _frequency(0),
      _packetIndex(0),
      _implicitHeaderMode(0),
      _onReceive(NULL),
      _onCadDone(NULL),
      _onTxDone(NULL)
{}

int LoRaClass::begin(long frequency)
{

```

```
  // setup pins
  gpio_init(_ss);
  gpio_set_dir(_ss, GPIO_OUT);
  // set SS high
  gpio_put(_ss, 1);

  if (_reset != -1) {
    gpio_init(_reset);
    gpio_set_dir(_reset, GPIO_OUT);

    // perform reset
    gpio_put(_reset, 0);
    sleep_ms(10);
    gpio_put(_reset, 1);
    sleep_ms(10);
  }


  // start SPI
  spi_init(SPI_PORT, LORA_DEFAULT_SPI_FREQUENCY);
  gpio_set_function(SX1278_MISO, GPIO_FUNC_SPI);
  gpio_set_function(SX1278_SCK, GPIO_FUNC_SPI);
  gpio_set_function(SX1278_MOSI, GPIO_FUNC_SPI);


  // Make the SPI pins available to picotool
  bi_decl(bi_3pins_with_func(SX1278_MISO, SX1278_MOSI, SX1278_SCK, GPIO_FUNC_SPI));

  gpio_init(SX1278_CS);
  gpio_set_dir(SX1278_CS, GPIO_OUT);
  gpio_put(SX1278_CS, 1);

  // Make the CS pin available to picotool
  bi_decl(bi_1pin_with_name(SX1278_CS, "SPI CS"));

  // check version
  uint8_t version = readRegister(REG_VERSION);
  if (version != 0x12) {
    return 0;
  }

  // put in sleep mode
  sleep();

  // set frequency
  setFrequency(frequency);

  // set base addresses
  writeRegister(REG_FIFO_TX_BASE_ADDR, 0);
  writeRegister(REG_FIFO_RX_BASE_ADDR, 0);

  // set LNA boost
  writeRegister(REG_LNA, readRegister(REG_LNA) | 0x03);

  // set auto AGC
  writeRegister(REG_MODEM_CONFIG_3, 0x04);

  // set output power to 17 dBm
  setTxPower(17);

  // put in standby mode
  idle();

  return 1;
}

void LoRaClass::end()
{
  // put in sleep mode
  sleep();

  // stop SPI
  spi_deinit(SPI_PORT);
}

int LoRaClass::beginPacket(int implicitHeader)
{
  if (isTransmitting()) {
    return 0;
  }

  // put in standby mode
  idle();

  if (implicitHeader) {
    implicitHeaderMode();
  } else {
```

```
    explicitHeaderMode();
  }

  // reset FIFO address and paload length
  writeRegister(REG_FIFO_ADDR_PTR, 0);
  writeRegister(REG_PAYLOAD_LENGTH, 0);

  return 1;
}

int LoRaClass::endPacket(bool async)
{

  if ((async) && (_onTxDone)) {
    writeRegister(REG_DIO_MAPPING_1, 0x40); // DIO0 => TXDONE
  }
  // put in TX mode
  writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_TX);

  if (!async) {
    // wait for TX done
    while ((readRegister(REG_IRQ_FLAGS) & IRQ_TX_DONE_MASK) == 0) {
      sleep_ms(0);
    }
    // clear IRQ's
    writeRegister(REG_IRQ_FLAGS, IRQ_TX_DONE_MASK);
  }

  return 1;
}

bool LoRaClass::isTransmitting()
{
  if ((readRegister(REG_OP_MODE) & MODE_TX) == MODE_TX) {
    return true;
  }

  if (readRegister(REG_IRQ_FLAGS) & IRQ_TX_DONE_MASK) {
    // clear IRQ's
    writeRegister(REG_IRQ_FLAGS, IRQ_TX_DONE_MASK);
  }

  return false;
}

int LoRaClass::parse_packet(int size)
{
  int packetLength = 0;

  int irqFlags = readRegister(REG_IRQ_FLAGS);

  if (size > 0) {
    implicitHeaderMode();
    writeRegister(REG_PAYLOAD_LENGTH, size & 0xff);
  } else {
    explicitHeaderMode();
  }

  // clear IRQ's
  writeRegister(REG_IRQ_FLAGS, irqFlags);
  writeRegister(REG_IRQ_FLAGS, irqFlags);

  if ((irqFlags & IRQ_RX_DONE_MASK) && (irqFlags & IRQ_PAYLOAD_CRC_ERROR_MASK) == 0) {
    // received a packet
    _packetIndex = 0;

    // read packet length
    if (_implicitHeaderMode) {
      packetLength = readRegister(REG_PAYLOAD_LENGTH);
    } else {
      packetLength = readRegister(REG_RX_NB_BYTES);
    }

    // set FIFO address to current RX address
    writeRegister(REG_FIFO_ADDR_PTR, readRegister(REG_FIFO_RX_CURRENT_ADDR));

    // put in standby mode
    idle();
  } else if (readRegister(REG_OP_MODE) != (MODE_LONG_RANGE_MODE | MODE_RX_SINGLE)) {
    // not currently in RX mode

    // reset FIFO address
    writeRegister(REG_FIFO_ADDR_PTR, 0);

    // put in single RX mode
    writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_RX_SINGLE);
  }
```

```
  return packetLength;
}

int LoRaClass::packetRssi()
{
  return (readRegister(REG_PKT_RSSI_VALUE) - (_frequency < RF_MID_BAND_THRESHOLD ? RSSI_OFFSET_LF_PORT : RSSI_OFFSET_HF_PORT));
}

float LoRaClass::packetSnr()
{
  return ((int8_t)readRegister(REG_PKT_SNR_VALUE)) * 0.25;
}

long LoRaClass::packetFrequencyError()
{
  int32_t freqError = 0;
  freqError = static_cast<int32_t>(readRegister(REG_FREQ_ERROR_MSB) & 0x111);
  freqError <<= 8L;
  freqError += static_cast<int32_t>(readRegister(REG_FREQ_ERROR_MID));
  freqError <<= 8L;
  freqError += static_cast<int32_t>(readRegister(REG_FREQ_ERROR_LSB));

  if (readRegister(REG_FREQ_ERROR_MSB) & 0x1000) { // Sign bit is on
    freqError -= 524288;                           // B1000'0000'0000'0000'0000
  }

  const float fXtal = 32E6; // FXOSC: crystal oscillator (XTAL) frequency (2.5. Chip Specification, p. 14)
  const float fError = ((static_cast<float>(freqError) * (1L << 24)) / fXtal) * (getSignalBandwidth() / 500000.0f); // p. 37

  return static_cast<long>(fError);
}

int LoRaClass::rssi()
{
  return (readRegister(REG_RSSI_VALUE) - (_frequency < RF_MID_BAND_THRESHOLD ? RSSI_OFFSET_LF_PORT : RSSI_OFFSET_HF_PORT));
}

size_t LoRaClass::write(uint8_t byte)
{
  return write(&byte, sizeof(byte));
}

size_t LoRaClass::write(const uint8_t *buffer, size_t size)
{
  int currentLength = readRegister(REG_PAYLOAD_LENGTH);

  // check size
  if ((currentLength + size) > MAX_PKT_LENGTH) {
    size = MAX_PKT_LENGTH - currentLength;
  }

  // write data
  for (size_t i = 0; i < size; i++) {
    writeRegister(REG_FIFO, buffer[i]);
  }

  // update length
  writeRegister(REG_PAYLOAD_LENGTH, currentLength + size);

  return size;
}

int LoRaClass::available()
{
  return (readRegister(REG_RX_NB_BYTES) - _packetIndex);
}

int LoRaClass::read()
{
  if (!available()) {
    return -1;
  }

  _packetIndex++;

  return readRegister(REG_FIFO);
}

int LoRaClass::peek()
{
  if (!available()) {
    return -1;
```

```
  }

  // store current FIFO address
  int currentAddress = readRegister(REG_FIFO_ADDR_PTR);

  // read
  uint8_t b = readRegister(REG_FIFO);

  // restore FIFO address
  writeRegister(REG_FIFO_ADDR_PTR, currentAddress);

  return b;
}

void LoRaClass::flush()
{
}

void LoRaClass::onReceive(void(*callback)(int))
{
  _onReceive = callback;

  if (callback) {
    gpio_set_irq_enabled_with_callback(_dio0, GPIO_IRQ_EDGE_RISE, true, &LoRaClass::onDio0Rise);
  } else {
    gpio_set_irq_enabled(_dio0, GPIO_IRQ_EDGE_RISE, false);
  }
}

void LoRaClass::onCadDone(void(*callback)(bool))
{
  _onCadDone = callback;

  if (callback) {
    gpio_set_irq_enabled_with_callback(_dio0, GPIO_IRQ_EDGE_RISE, true,
                                       &LoRaClass::onDio0Rise);
  } else {
    gpio_set_irq_enabled(_dio0, GPIO_IRQ_EDGE_RISE, false);
  }
}

void LoRaClass::onTxDone(void (*callback)())
{
  _onTxDone = callback;

  if (callback) {
    gpio_set_irq_enabled_with_callback(_dio0, GPIO_IRQ_EDGE_RISE, true, &LoRaClass::onDio0Rise);
  } else {
    gpio_set_irq_enabled(_dio0, GPIO_IRQ_EDGE_RISE, false);
  }
}

void LoRaClass::receive(int size)
{

  writeRegister(REG_DIO_MAPPING_1, 0x00); // DIO0 => RXDONE

  if (size > 0) {
    implicitHeaderMode();

    writeRegister(REG_PAYLOAD_LENGTH, size & 0xff);
  } else {
    explicitHeaderMode();
  }

  writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_RX_CONTINUOUS);
}

void LoRaClass::channelActivityDetection(void)
{
  writeRegister(REG_DIO_MAPPING_1, 0x80); // DIO0 => CADDONE
  writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_CAD);
}

void LoRaClass::idle()
{
  writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_STDBY);
}

void LoRaClass::sleep()
{
  writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_SLEEP);
}

void LoRaClass::setTxPower(int level, int outputPin)
{
  if (PA_OUTPUT_RFO_PIN == outputPin) {
```

```
    // RFO
    if (level < 0) {
      level = 0;
    } else if (level > 14) {
      level = 14;
    }

    writeRegister(REG_PA_CONFIG, 0x70 | level);
  } else {
    // PA BOOST
    if (level > 17) {
      if (level > 20) {
        level = 20;
      }

      // subtract 3 from level, so 18 - 20 maps to 15 - 17
      level -= 3;

      // High Power +20 dBm Operation (Semtech SX1276/77/78/79 5.4.3.)
      writeRegister(REG_PA_DAC, 0x87);
      setOCP(140);
    } else {
      if (level < 2) {
        level = 2;
      }
      //Default value PA_HF/LF or +17dBm
      writeRegister(REG_PA_DAC, 0x84);
      setOCP(100);
    }

    writeRegister(REG_PA_CONFIG, PA_BOOST | (level - 2));
  }
}

void LoRaClass::setFrequency(long frequency)
{
  _frequency = frequency;

  uint64_t frf = ((uint64_t)frequency << 19) / 32000000;

  writeRegister(REG_FRF_MSB, (uint8_t)(frf >> 16));
  writeRegister(REG_FRF_MID, (uint8_t)(frf >> 8));
  writeRegister(REG_FRF_LSB, (uint8_t)(frf >> 0));
}

int LoRaClass::getSpreadingFactor()
{
  return readRegister(REG_MODEM_CONFIG_2) >> 4;
}

void LoRaClass::setSpreadingFactor(int sf)
{
  if (sf < 6) {
    sf = 6;
  } else if (sf > 12) {
    sf = 12;
  }

  if (sf == 6) {
    writeRegister(REG_DETECTION_OPTIMIZE, 0xc5);
    writeRegister(REG_DETECTION_THRESHOLD, 0x0c);
  } else {
    writeRegister(REG_DETECTION_OPTIMIZE, 0xc3);
    writeRegister(REG_DETECTION_THRESHOLD, 0x0a);
  }

  writeRegister(REG_MODEM_CONFIG_2, (readRegister(REG_MODEM_CONFIG_2) & 0x0f) | ((sf << 4) & 0xf0));
  setLdoFlag();
}

long LoRaClass::getSignalBandwidth()
{
  uint8_t bw = (readRegister(REG_MODEM_CONFIG_1) >> 4);

  switch (bw) {
  case 0: return 7.8E3;
  case 1: return 10.4E3;
  case 2: return 15.6E3;
  case 3: return 20.8E3;
  case 4: return 31.25E3;
  case 5: return 41.7E3;
  case 6: return 62.5E3;
  case 7: return 125E3;
  case 8: return 250E3;
  case 9: return 500E3;
  }

```

```
  return -1;
}

void LoRaClass::setSignalBandwidth(long sbw)
{
  int bw;

  if (sbw <= 7.8E3) {
    bw = 0;
  } else if (sbw <= 10.4E3) {
    bw = 1;
  } else if (sbw <= 15.6E3) {
    bw = 2;
  } else if (sbw <= 20.8E3) {
    bw = 3;
  } else if (sbw <= 31.25E3) {
    bw = 4;
  } else if (sbw <= 41.7E3) {
    bw = 5;
  } else if (sbw <= 62.5E3) {
    bw = 6;
  } else if (sbw <= 125E3) {
    bw = 7;
  } else if (sbw <= 250E3) {
    bw = 8;
  } else /*if (sbw <= 250E3)*/ {
    bw = 9;
  }

  writeRegister(REG_MODEM_CONFIG_1, (readRegister(REG_MODEM_CONFIG_1) & 0x0f) | (bw << 4));
  setLdoFlag();
}

void LoRaClass::setLdoFlag()
{
  long symbolDuration = 1000 / ( getSignalBandwidth() / (1L << getSpreadingFactor()) ) ;

  bool ldoOn = symbolDuration > 16;

  uint8_t config3 = readRegister(REG_MODEM_CONFIG_3);

  config3 = ldoOn ? config3 | (1 << 3) : config3 & ~(1 << 3);

  writeRegister(REG_MODEM_CONFIG_3, config3);
}

void LoRaClass::setCodingRate4(int denominator)
{
  if (denominator < 5) {
    denominator = 5;
  } else if (denominator > 8) {
    denominator = 8;
  }

  int cr = denominator - 4;

  writeRegister(REG_MODEM_CONFIG_1, (readRegister(REG_MODEM_CONFIG_1) & 0xf1) | (cr << 1));
}

void LoRaClass::setPreambleLength(long length)
{
  writeRegister(REG_PREAMBLE_MSB, (uint8_t)(length >> 8));
  writeRegister(REG_PREAMBLE_LSB, (uint8_t)(length >> 0));
}

void LoRaClass::setSyncWord(int sw)
{
  writeRegister(REG_SYNC_WORD, sw);
}

void LoRaClass::enableCrc()
{
  writeRegister(REG_MODEM_CONFIG_2, readRegister(REG_MODEM_CONFIG_2) | 0x04);
}

void LoRaClass::disableCrc()
{
  writeRegister(REG_MODEM_CONFIG_2, readRegister(REG_MODEM_CONFIG_2) & 0xfb);
}

void LoRaClass::enableInvertIQ()
{
  writeRegister(REG_INVERTIQ,  0x66);
  writeRegister(REG_INVERTIQ2, 0x19);
}

void LoRaClass::disableInvertIQ()
```

```
{
  writeRegister(REG_INVERTIQ,  0x27);
  writeRegister(REG_INVERTIQ2, 0x1d);
}

void LoRaClass::setOCP(uint8_t mA)
{
  uint8_t ocpTrim = 27;

  if (mA <= 120) {
    ocpTrim = (mA - 45) / 5;
  } else if (mA <= 240) {
    ocpTrim = (mA + 30) / 10;
  }

  writeRegister(REG_OCP, 0x20 | (0x1F & ocpTrim));
}

void LoRaClass::setGain(uint8_t gain)
{
  // check allowed range
  if (gain > 6) {
    gain = 6;
  }

  // set to standby
  idle();

  // set gain
  if (gain == 0) {
    // if gain = 0, enable AGC
    writeRegister(REG_MODEM_CONFIG_3, 0x04);
  } else {
    // disable AGC
    writeRegister(REG_MODEM_CONFIG_3, 0x00);

    // clear Gain and set LNA boost
    writeRegister(REG_LNA, 0x03);

    // set gain
    writeRegister(REG_LNA, readRegister(REG_LNA) | (gain « 5));
  }
}

uint8_t LoRaClass::random()
{
  return readRegister(REG_RSSI_WIDEBAND);
}

void LoRaClass::set_pins(int ss, int reset, int dio0)
{
  _ss = ss;
  _reset = reset;
  _dio0 = dio0;
}

void LoRaClass::setSPI(spi_inst_t& spi)
{
  _spi = &spi;
}

void LoRaClass::setSPIFrequency(uint32_t frequency)
{
  spi_set_baudrate(SPI_PORT, frequency);
}

void LoRaClass::dumpRegisters()
{
  for (int i = 0; i < 128; i++) {
    printf("0x%x: 0x%x\n", i, readRegister(i));
  }
}

void LoRaClass::explicitHeaderMode()
{
  _implicitHeaderMode = 0;

  writeRegister(REG_MODEM_CONFIG_1, readRegister(REG_MODEM_CONFIG_1) & 0xfe);
}

void LoRaClass::implicitHeaderMode()
{
  _implicitHeaderMode = 1;

  writeRegister(REG_MODEM_CONFIG_1, readRegister(REG_MODEM_CONFIG_1) | 0x01);
}

```

```
void LoRaClass::handleDio0Rise()
{
  int irqFlags = readRegister(REG_IRQ_FLAGS);

  // clear IRQ's
  writeRegister(REG_IRQ_FLAGS, irqFlags);
  writeRegister(REG_IRQ_FLAGS, irqFlags);

  if ((irqFlags & IRQ_CAD_DONE_MASK) != 0) {
    if (_onCadDone) {
      _onCadDone((irqFlags & IRQ_CAD_DETECTED_MASK) != 0);
    }
  } else if ((irqFlags & IRQ_PAYLOAD_CRC_ERROR_MASK) == 0) {

    if ((irqFlags & IRQ_RX_DONE_MASK) != 0) {
      // received a packet
      _packetIndex = 0;

      // read packet length
      int packetLength = _implicitHeaderMode ? readRegister(REG_PAYLOAD_LENGTH) :
    readRegister(REG_RX_NB_BYTES);

      // set FIFO address to current RX address
      writeRegister(REG_FIFO_ADDR_PTR, readRegister(REG_FIFO_RX_CURRENT_ADDR));

      if (_onReceive) {
        _onReceive(packetLength);
      }
    } else if ((irqFlags & IRQ_TX_DONE_MASK) != 0) {
      if (_onTxDone) {
        _onTxDone();
      }
    }
  }
}

uint8_t LoRaClass::readRegister(uint8_t address)
{
  return singleTransfer(address & 0x7f, 0x00);
}

void LoRaClass::writeRegister(uint8_t address, uint8_t value)
{
  singleTransfer(address | 0x80, value);
}

uint8_t LoRaClass::singleTransfer(uint8_t address, uint8_t value)
{
  uint8_t response;

  gpio_put(_ss, 0);

  spi_write_blocking(SPI_PORT, &address, 1);
  spi_write_read_blocking(SPI_PORT, &value, &response, 1);

  gpio_put(_ss, 1);

  return response;
}

void LoRaClass::onDio0Rise(uint gpio, uint32_t events)
{
  gpio_acknowledge_irq(gpio, events);
  LoRa.handleDio0Rise();
}

LoRaClass LoRa;
```

8.37 lib/comms/LoRa/LoRa-RP2040.h File Reference

```
#include "pico/stdlib.h"
#include "pico/binary_info.h"
#include "hardware/gpio.h"
#include "hardware/spi.h"
#include "string.h"
#include "Print.h"
```

```
#include "pin_config.h"
```

Include dependency graph for LoRa-RP2040.h:

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

Classes

- class LoRaClass

Functions

- static void __empty ()

Variables

- LoRaClass LoRa

8.37.1 Function Documentation

8.37.1.1 __empty()

```
static void __empty ()  [static]
```

8.37.2 Variable Documentation

8.37.2.1 LoRa

```
LoRaClass LoRa  [extern]
```

Definition at line 753 of file LoRa-RP2040.cpp.

8.38 LoRa-RP2040.h

Go to the documentation of this file.

```
#ifndef LORA_H
#define LORA_H

#include "pico/stdlib.h"
#include "pico/binary_info.h"
#include "hardware/gpio.h"
#include "hardware/spi.h"
#include "string.h"
#include "Print.h"
#include "pin_config.h"

#endif

static void __empty();

//class LoRaClass : public Stream {
class LoRaClass : public Print {
public:
  LoRaClass();

  int begin(long frequency);
  void end();

  int beginPacket(int implicitHeader = false);
  int endPacket(bool async = false);

  int parse_packet(int size = 0);
  int packetRssi();
  float packetSnr();
  long packetFrequencyError();

  int rssi();

  // from Print
  virtual size_t write(uint8_t byte);
  virtual size_t write(const uint8_t *buffer, size_t size);

  // from Stream
  virtual int available();
  virtual int read();
  virtual int peek();
  virtual void flush();

  void onCadDone(void (*callback)(bool));
  void onReceive(void (*callback)(int));
  void onTxDone(void (*callback)());

  void receive(int size = 0);
  void channelActivityDetection(void);

  void idle();
  void sleep();

  // size_t print(const char* c);

  void setTxPower(int level, int outputPin = PA_OUTPUT_PA_BOOST_PIN);
  void setFrequency(long frequency);
  void setSpreadingFactor(int sf);
  void setSignalBandwidth(long sbw);
  void setCodingRate4(int denominator);
  void setPreambleLength(long length);
  void setSyncWord(int sw);
  void enableCrc();
  void disableCrc();
  void enableInvertIQ();
  void disableInvertIQ();

  void setOCP(uint8_t mA); // Over Current Protection control

  void setGain(uint8_t gain); // Set LNA gain

  // deprecated
  void crc() { enableCrc(); }
  void noCrc() { disableCrc(); }

  uint8_t random();

  void set_pins(int ss = LORA_DEFAULT_SS_PIN, int reset = LORA_DEFAULT_RESET_PIN, int dio0 =
    LORA_DEFAULT_DIO0_PIN);
  void setSPI(spi_inst_t &spi);
  void setSPIFrequency(uint32_t frequency);

```

```
  void dumpRegisters();

private:
  void explicitHeaderMode();
  void implicitHeaderMode();

  void handleDio0Rise();
  bool isTransmitting();

  int getSpreadingFactor();
  long getSignalBandwidth();

  void setLdoFlag();

  uint8_t readRegister(uint8_t address);
  void writeRegister(uint8_t address, uint8_t value);
  uint8_t singleTransfer(uint8_t address, uint8_t value);

  static void onDio0Rise(uint, uint32_t);

private:
  // SPISettings _spiSettings;
  spi_inst_t *_spi;
  int _ss;
  int _reset;
  int _dio0;
  long _frequency;
  int _packetIndex;
  int _implicitHeaderMode;
  void (*_onReceive)(int);
  void (*_onCadDone)(bool);
  void (*_onTxDone)();
};

extern LoRaClass LoRa;
```

8.39 lib/comms/LoRa/Print.cpp File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <math.h>
#include "Print.h"
```

Include dependency graph for Print.cpp:

8.40 Print.cpp

Go to the documentation of this file.

```
/*
  Copyright (c) 2014 Arduino.  All right reserved.

  This library is free software; you can redistribute it and/or
  modify it under the terms of the GNU Lesser General Public
  License as published by the Free Software Foundation; either
  version 2.1 of the License, or (at your option) any later version.

  This library is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
  See the GNU Lesser General Public License for more details.

  You should have received a copy of the GNU Lesser General Public
  License along with this library; if not, write to the Free Software
  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
*/

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <string>
#include <math.h>

#include "Print.h"

using std::string;
// Public Methods //////////////////////////////////////////////////////////////

/* default implementation: may be overridden */
size_t Print::write(const uint8_t *buffer, size_t size)
{
  size_t n = 0;
  while (size--) {
    if (write(*buffer++)) n++;
    else break;
  }
  return n;
}

size_t Print::print(string str)
{
  return write(str.c_str(), str.size());
}

// size_t Print::print(const char str[])
// {
//   return write(str);
// }

size_t Print::print(char c)
{
  return write(c);
}

size_t Print::print(const char* c){
   return write(c);
}


size_t Print::print(unsigned char b, int base)
{
  return print((unsigned long) b, base);
}

size_t Print::print(int n, int base)
{
  return print((long) n, base);
}

size_t Print::print(unsigned int n, int base)
{
  return print((unsigned long) n, base);
}

size_t Print::print(long n, int base)
{
  if (base == 0) {
    return write(n);
  } else if (base == 10) {
    if (n < 0) {
      int t = print('-');
      n = -n;
      return printNumber(n, 10) + t;
    }
    return printNumber(n, 10);
  } else {
```

```
    return printNumber(n, base);
  }
}

size_t Print::print(unsigned long n, int base)
{
  if (base == 0) return write(n);
  else return printNumber(n, base);
}

size_t Print::print(long long n, int base)
{
  if (base == 0) {
    return write(n);
  } else if (base == 10) {
    if (n < 0) {
      int t = print('-');
      n = -n;
      return printULLNumber(n, 10) + t;
    }
    return printULLNumber(n, 10);
  } else {
    return printULLNumber(n, base);
  }
}

size_t Print::print(unsigned long long n, int base)
{
  if (base == 0) return write(n);
  else return printULLNumber(n, base);
}

size_t Print::print(double n, int digits)
{
  return printFloat(n, digits);
}

size_t Print::println(void)
{
  return write("\r\n");
}

size_t Print::println(const char c[])
{
  size_t n = print(c);
  n += println();
  return n;
}

size_t Print::println(char c)
{
  size_t n = print(c);
  n += println();
  return n;
}

size_t Print::println(unsigned char b, int base)
{
  size_t n = print(b, base);
  n += println();
  return n;
}

size_t Print::println(int num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(unsigned int num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(long num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(unsigned long num, int base)
{
  size_t n = print(num, base);
```

```
  n += println();
  return n;
}

size_t Print::println(long long num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(unsigned long long num, int base)
{
  size_t n = print(num, base);
  n += println();
  return n;
}

size_t Print::println(double num, int digits)
{
  size_t n = print(num, digits);
  n += println();
  return n;
}


// Private Methods /////////////////////////////////////////////////////////////

size_t Print::printNumber(unsigned long n, uint8_t base)
{
  char buf[8 * sizeof(long) + 1]; // Assumes 8-bit chars plus zero byte.
  char *str = &buf[sizeof(buf) - 1];

  *str = '\0';

  // prevent crash if called with base == 1
  if (base < 2) base = 10;

  do {
    char c = n % base;
    n /= base;

    *--str = c < 10 ? c + '0' : c + 'A' - 10;
  } while(n);

  return write(str);
}

// REFERENCE IMPLEMENTATION FOR ULL
// size_t Print::printULLNumber(unsigned long long n, uint8_t base)
// {
  // // if limited to base 10 and 16 the bufsize can be smaller
  // char buf[65];
  // char *str = &buf[64];

  // *str = '\0';

  // // prevent crash if called with base == 1
  // if (base < 2) base = 10;

    // do {
      // unsigned long long t = n / base;
      // char c = n - t * base;  // faster than c = n%base;
      // n = t;
      // *--str = c < 10 ? c + '0' : c + 'A' - 10;
  // } while(n);

  // return write(str);
// }

// FAST IMPLEMENTATION FOR ULL
size_t Print::printULLNumber(unsigned long long n64, uint8_t base)
{
  // if limited to base 10 and 16 the bufsize can be 20
  char buf[64];
  uint8_t i = 0;
  uint8_t innerLoops = 0;

  // prevent crash if called with base == 1
  if (base < 2) base = 10;

  // process chunks that fit in "16 bit math".
  uint16_t top = 0xFFFF / base;
  uint16_t th16 = 1;
  while (th16 < top)
  {
    th16 *= base;
```

```
    innerLoops++;
  }

  while (n64 > th16)
  {
    // 64 bit math part
    uint64_t q = n64 / th16;
    uint16_t r = n64 - q*th16;
    n64 = q;

    // 16 bit math loop to do remainder. (note buffer is filled reverse)
    for (uint8_t j=0; j < innerLoops; j++)
    {
      uint16_t qq = r/base;
      buf[i++] = r - qq*base;
      r = qq;
    }
  }

  uint16_t n16 = n64;
  while (n16 > 0)
  {
    uint16_t qq = n16/base;
    buf[i++] = n16 - qq*base;
    n16 = qq;
  }

  size_t bytes = i;
  for (; i > 0; i--)
    write((char) (buf[i - 1] < 10 ?
    '0' + buf[i - 1] :
    'A' + buf[i - 1] - 10));

  return bytes;
}

size_t Print::printFloat(double number, int digits)
{
  if (digits < 0)
    digits = 2;

  size_t n = 0;

  if (isnan(number)) return print("nan");
  if (isinf(number)) return print("inf");
  if (number > 4294967040.0) return print ("ovf");  // constant determined empirically
  if (number <-4294967040.0) return print ("ovf");  // constant determined empirically

  // Handle negative numbers
  if (number < 0.0)
  {
     n += print('-');
     number = -number;
  }

  // Round correctly so that print(1.999, 2) prints as "2.00"
  double rounding = 0.5;
  for (uint8_t i=0; i<digits; ++i)
    rounding /= 10.0;

  number += rounding;

  // Extract the integer part of the number and print it
  unsigned long int_part = (unsigned long)number;
  double remainder = number - (double)int_part;
  n += print(int_part);

  // Print the decimal point, but only if there are digits beyond
  if (digits > 0) {
    n += print(".");
  }

  // Extract digits from the remainder one at a time
  while (digits-- > 0)
  {
    remainder *= 10.0;
    unsigned int toPrint = (unsigned int)remainder;
    n += print(toPrint);
    remainder -= toPrint;
  }

  return n;
}
```

8.41 lib/comms/LoRa/Print.h File Reference

```
#include <inttypes.h>
#include <stdio.h>
#include <string>
```

Include dependency graph for Print.h:

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

Classes

- class Print

Macros

- #define DEC 10
- #define HEX 16
- #define OCT 8
- #define BIN 2

8.41.1 Macro Definition Documentation

8.41.1.1 DEC

```
#define DEC 10
```

Definition at line 26 of file Print.h.

8.41.1.2 HEX

```
#define HEX 16
```

Definition at line 27 of file Print.h.

8.41.1.3 OCT

```
#define OCT 8
```

Definition at line 28 of file Print.h.

8.41.1.4 BIN

```
#define BIN 2
```

Definition at line 29 of file Print.h.

8.42 Print.h

Go to the documentation of this file.

```
00001 /*
00002   Copyright (c) 2016 Arduino LLC.  All right reserved.
00003
00004   This library is free software; you can redistribute it and/or
00005   modify it under the terms of the GNU Lesser General Public
00006   License as published by the Free Software Foundation; either
00007   version 2.1 of the License, or (at your option) any later version.
00008
00009   This library is distributed in the hope that it will be useful,
00010   but WITHOUT ANY WARRANTY; without even the implied warranty of
00011   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
00012   See the GNU Lesser General Public License for more details.
00013
00014   You should have received a copy of the GNU Lesser General Public
00015   License along with this library; if not, write to the Free Software
00016   Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
00017 */
00018
00019 #pragma once
00020
00021 #include <inttypes.h>
00022 #include <stdio.h> // for size_t
00023 #include <string>
00024
00025
00026 #define DEC 10
00027 #define HEX 16
00028 #define OCT 8
00029 #define BIN 2
00030
00031
00032 using std::string;
00033
00034 class Print
00035 {
00036   private:
00037     int write_error;
00038     size_t printNumber(unsigned long, uint8_t);
00039     size_t printULLNumber(unsigned long long, uint8_t);
00040     size_t printFloat(double, int);
00041   protected:
00042     void setWriteError(int err = 1) { write_error = err; }
00043   public:
00044     Print() : write_error(0) {}
00045
00046     int getWriteError() { return write_error; }
```

```
    void clearWriteError() { setWriteError(0); }

    virtual size_t write(uint8_t) = 0;
    size_t write(const char *str) {
      if (str == NULL) return 0;
      return write((const uint8_t *)str, strlen(str));
    }
    virtual size_t write(const uint8_t *buffer, size_t size);
    size_t write(const char *buffer, size_t size) {
      return write((const uint8_t *)buffer, size);
    }

    // default to zero, meaning "a single write may block"
    // should be overridden by subclasses with buffering
    virtual int availableForWrite() { return 0; }

   //  size_t print(const char[]);
    size_t print(char);
    size_t print(const char*);
    size_t print(string c);
    size_t print(unsigned char, int = DEC);
    size_t print(int, int = DEC);
    size_t print(unsigned int, int = DEC);
    size_t print(long, int = DEC);
    size_t print(unsigned long, int = DEC);
    size_t print(long long, int = DEC);
    size_t print(unsigned long long, int = DEC);
    size_t print(double, int = 2);

    size_t println(const char[]);
    size_t println(char);
    size_t println(unsigned char, int = DEC);
    size_t println(int, int = DEC);
    size_t println(unsigned int, int = DEC);
    size_t println(long, int = DEC);
    size_t println(unsigned long, int = DEC);
    size_t println(long long, int = DEC);
    size_t println(unsigned long long, int = DEC);
    size_t println(double, int = 2);
    size_t println(void);

    virtual void flush() { /* Empty implementation for backward compatibility */ }
};
```

8.43 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

Enumerations

- enum class ExecutionResult { SUCCESS , ERROR , INFO }
- enum class OperationType {
  GET , SET , ANS , ERR ,
  INF }
- 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 }

Functions

- std::string exception_type_to_string (ExceptionType type)

  Converts an ExceptionType to a string.
- std::string operation_type_to_string (OperationType type)

  Converts an OperationType to a string.
- OperationType string_to_operation_type (const std::string &str)

  Converts a string to an OperationType.
- std::vector< uint8_t > hex_string_to_bytes (const std::string &hexString)

  Converts a hex string to a vector of bytes.
- std::string value_unit_type_to_string (ValueUnit unit)

  Converts a ValueUnit to a string.

Variables

- const std::string FRAME_BEGIN = "KBST"
- const std::string FRAME_END = "TSBK"
- const char DELIMITER = ';'

8.43.1 Enumeration Type Documentation

8.43.1.1 ExecutionResult

```
enum class ExecutionResult  [strong]
```

Enumerator

SUCCESS	
ERROR	
INFO	

Definition at line 26 of file protocol.h.

8.43.1.2 OperationType

```
enum class OperationType [strong]
```

Enumerator

GET	
SET	
ANS	
ERR	
INF	

Definition at line 32 of file protocol.h.

8.43.1.3 CommandAccessLevel

```
enum class CommandAccessLevel [strong]
```

Enumerator

NONE	
READ_ONLY	
WRITE_ONLY	
READ_WRITE	

Definition at line 40 of file protocol.h.

8.43.1.4 ValueUnit

```
enum class ValueUnit [strong]
```

Enumerator

UNDEFINED	
SECOND	
VOLT	
BOOL	
DATETIME	
TEXT	
MILIAMP	

Definition at line 47 of file protocol.h.

8.43.1.5 ExceptionType

```
enum class ExceptionType [strong]
```

Enumerator

NONE	
NOT_ALLOWED	
INVALID_PARAM	
INVALID_OPERATION	
PARAM_UNECESSARY	

Definition at line 57 of file protocol.h.

8.43.1.6 Interface

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

Enumerator

UART	
LORA	

Definition at line 65 of file protocol.h.

8.43.2 Function Documentation

8.43.2.1 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:

8.43.2.2 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:

8.43.2.3 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 67 of file utils_converters.cpp.

Here is the caller graph for this function:

8.43.2.4 hex_string_to_bytes()

```
std::vector< uint8_t > hex_string_to_bytes (
            const std::string & hexString)
```

Converts a hex string to a vector of bytes.

Parameters

hexString	The hex string to convert.

Returns

A vector of bytes representing the hex string.

Definition at line 81 of file utils_converters.cpp.

Here is the caller graph for this function:

8.43.2.5 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:

8.43.3 Variable Documentation

8.43.3.1 FRAME_BEGIN

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

Definition at line 22 of file protocol.h.

8.43.3.2 FRAME_END

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

Definition at line 23 of file protocol.h.

8.43.3.3 DELIMITER

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

Definition at line 24 of file protocol.h.

8.44 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
00022 const std::string FRAME_BEGIN = "KBST";
00023 const std::string FRAME_END = "TSBK";
00024 const char DELIMITER = ';';
00025
00026 enum class ExecutionResult {
00027     SUCCESS,
00028     ERROR,
00029     INFO
00030 };
00031
00032 enum class OperationType {
00033     GET,
00034     SET,
00035     ANS,
00036     ERR,
00037     INF
00038 };
00039
00040 enum class CommandAccessLevel {
00041     NONE,
00042     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 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
```

8.45 lib/comms/receive.cpp File Reference

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

`#include "communication.h"`

Include dependency graph for receive.cpp:

Functions

- void on_receive (int packetSize)

  Callback function for handling received LoRa packets.
- void handle_uart_input ()

  Handles UART input.

8.45.1 Detailed Description

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

Definition in file receive.cpp.

8.45.2 Function Documentation

8.45.2.1 on_receive()

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

Callback function for handling received LoRa packets.

Parameters

packetSize	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:

8.45.2.2 handle_uart_input()

```
void handle_uart_input ()
```

Handles UART input.

Reads characters from the UART port, appends them to a buffer, and processes the buffer when a newline character is received.

Definition at line 76 of file receive.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.46 receive.cpp

Go to the documentation of this file.

```
#include "communication.h"



void on_receive(int packetSize) {
    if (packetSize == 0) return;

    uint8_t buffer[256];
    int bytesRead = 0;

    while (LoRa.available() && bytesRead < packetSize) {
        buffer[bytesRead++] = LoRa.read();
    }

    // Extract LoRa metadata
    uint8_t receivedDestination = buffer[0];
    uint8_t receivedLocalAddress = buffer[1];

    // Validate metadata (optional, for security)
    if (receivedDestination != lora_address_local) {
        uart_print("Error: Destination address mismatch!", VerbosityLevel::ERROR);
        return;
    }

```

```
    if (receivedLocalAddress != lora_address_remote) {
        uart_print("Error: Local address mismatch!", VerbosityLevel::ERROR);
        return;
    }

    // Find the starting index of the actual frame data
    int startIndex = 2; // Start after the metadata

    // Extract the frame data
    std::string received = std::string(reinterpret_cast<char*>(buffer + startIndex), bytesRead - startIndex);

    if (received.empty()) return;

    // Debug: Print raw hex values
    std::stringstream hexDump;
    hexDump « "Raw bytes: ";
    for (int i = 0; i < bytesRead; i++) {
        hexDump « std::hex « std::setfill('0') « std::setw(2)
                « static_cast<int>(buffer[i]) « " ";
    }
    uart_print(hexDump.str(), VerbosityLevel::DEBUG);

    // Find frame boundaries
    size_t headerPos = received.find(FRAME_BEGIN);
    size_t footerPos = received.find(FRAME_END);

    if (headerPos != std::string::npos && footerPos != std::string::npos && footerPos > headerPos) {
        std::string frameData = received.substr(headerPos, footerPos + FRAME_END.length() - headerPos);
        uart_print("Extracted frame (length=" + std::to_string(frameData.length()) + "): " + frameData, VerbosityLevel::DEBUG);
        frame_process(frameData, Interface::LORA);
    } else {
        uart_print("No valid frame found in received data", VerbosityLevel::WARNING);
    }
}


void handle_uart_input() {
    static std::string uartBuffer;

    while (uart_is_readable(DEBUG_UART_PORT)) {
        char c = uart_getc(DEBUG_UART_PORT);

        if (c == '\r' || c == '\n') {
            uart_print("Received UART string: " + uartBuffer, VerbosityLevel::DEBUG);
            frame_process(uartBuffer, Interface::UART);
            uartBuffer.clear();
        } else {
            uartBuffer += c;
        }
    }
}
```

8.47 lib/comms/send.cpp File Reference

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

`#include "communication.h"`

Include dependency graph for send.cpp:

Functions

- void send_message (string outgoing)

Sends a message using LoRa.

- void send_frame_lora (const Frame &frame)
- void send_frame_uart (const Frame &frame)
- void send_frame (const Frame &frame)
- void split_and_send_message (const uint8_t *data, size_t length)

  Sends a large packet using LoRa.

8.47.1 Detailed Description

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

Definition in file send.cpp.

8.47.2 Function Documentation

8.47.2.1 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:

8.47.2.2 send_frame_lora()

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

Definition at line 37 of file send.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.47.2.3 send_frame_uart()

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

Definition at line 42 of file send.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.47.2.4 send_frame()

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

Definition at line 48 of file send.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.47.2.5 split_and_send_message()

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

Sends a large packet using LoRa.

Parameters

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.

8.48 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());

    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

    std::string messageToLog = "Sent message of size " + std::to_string(n);
    messageToLog += " to 0x" + std::to_string(lora_address_remote);
```

```
    messageToLog += " containing: " + string(send);

    uart_print(messageToLog, VerbosityLevel::DEBUG);

    LoRa.flush();
}


void send_frame_lora(const Frame& frame) {
    std::string encodedFrame = frame_encode(frame);
    send_message(encodedFrame);
}

void send_frame_uart(const Frame& frame) {
    std::string encodedFrame = frame_encode(frame);
    uart_print(encodedFrame);
}

[[deprecated("Use send_frame_lora or send_frame_uart instead")]]
void send_frame(const Frame& frame) {
    send_frame_lora(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 chunkSize = ((length - offset) < MAX_PKT_SIZE) ? (length - offset) : MAX_PKT_SIZE;
        LoRa.beginPacket();
        LoRa.write(&data[offset], chunkSize);
        LoRa.endPacket();
        offset += chunkSize;
        sleep_ms(100);
    }
}

```

8.49 lib/comms/utils_converters.cpp File Reference

Implements utility functions for converting between different data types.

`#include "communication.h"`

Include dependency graph for utils_converters.cpp:

Functions

- std::string exception_type_to_string (ExceptionType type)

  Converts an ExceptionType to a string.
- std::string value_unit_type_to_string (ValueUnit unit)

  Converts a ValueUnit to a string.
- std::string operation_type_to_string (OperationType type)

  Converts an OperationType to a string.
- OperationType string_to_operation_type (const std::string &str)

  Converts a string to an OperationType.
- std::vector< uint8_t > hex_string_to_bytes (const std::string &hexString)

  Converts a hex string to a vector of bytes.

8.49.1 Detailed Description

Implements utility functions for converting between different data types.

Definition in file utils_converters.cpp.

8.49.2 Function Documentation

8.49.2.1 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:

8.49.2.2 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:

8.49.2.3 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:

8.49.2.4 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 67 of file utils_converters.cpp.

Here is the caller graph for this function:

8.49.2.5 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 81 of file utils_converters.cpp.

Here is the caller graph for this function:

8.50 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::ANS: return "ANS";
        case OperationType::ERR: return "ERR";
        case OperationType::INF: return "INF";
        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 == "ANS") return OperationType::ANS;
    if (str == "ERR") return OperationType::ERR;
    if (str == "INF") return OperationType::INF;
    return OperationType::GET; // Default to GET
}

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;
}
```

8.51 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:

Functions

- void check_power_events (PowerManager &pm)

  Checks power statuses and triggers events based on voltage trends.

Variables

- volatile uint16_t eventLogId = 0

  Global event log ID counter.
- static PowerEvent lastPowerState = PowerEvent::LOW_BATTERY

  Stores the last known power state.
- static constexpr float FALL_RATE_THRESHOLD = -0.02f

  Threshold for detecting a falling voltage rate.
- static constexpr int FALLING_TREND_REQUIRED = 3

  Number of consecutive falling voltage readings required to trigger a power falling event.
- static constexpr float VOLTAGE_LOW_THRESHOLD = 4.7f

  Voltage threshold for detecting a low battery condition.
- static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f

  Voltage threshold for detecting an overcharge condition.
- static int fallingTrendCount = 0

  Counter for consecutive falling voltage readings.
- bool lastSolarState = false

  Stores the last known solar charging state.
- bool lastUSBState = false

  Stores the last known USB connection state.
- DS3231 systemClock

  External declaration of the system clock.
- EventManagerImpl eventManager

  Global instance of the EventManager implementation.

8.51.1 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.

8.51.2 Function Documentation

8.51.2.1 check_power_events()

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

Checks power statuses and triggers events based on voltage trends.

Parameters

pm	Reference to the PowerManager object.

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

Definition at line 139 of file event_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.51.3 Variable Documentation

8.51.3.1 eventLogId

```
volatile uint16_t eventLogId = 0
```

Global event log ID counter.

Definition at line 20 of file event_manager.cpp.

8.51.3.2 lastPowerState

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

Stores the last known power state.

Definition at line 25 of file event_manager.cpp.

8.51.3.3 FALL_RATE_THRESHOLD

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

Threshold for detecting a falling voltage rate.

Definition at line 30 of file event_manager.cpp.

8.51.3.4 FALLING_TREND_REQUIRED

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

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

Definition at line 35 of file event_manager.cpp.

8.51.3.5 VOLTAGE_LOW_THRESHOLD

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

Voltage threshold for detecting a low battery condition.

Definition at line 40 of file event_manager.cpp.

8.51.3.6 VOLTAGE_OVERCHARGE_THRESHOLD

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

Voltage threshold for detecting an overcharge condition.

Definition at line 45 of file event_manager.cpp.

8.51.3.7 fallingTrendCount

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

Counter for consecutive falling voltage readings.

Definition at line 50 of file event_manager.cpp.

8.51.3.8 lastSolarState

```
bool lastSolarState = false
```

Stores the last known solar charging state.

Definition at line 55 of file event_manager.cpp.

8.51.3.9 lastUSBState

```
bool lastUSBState = false
```

Stores the last known USB connection state.

Definition at line 60 of file event_manager.cpp.

8.51.3.10 systemClock

```
DS3231 systemClock  [extern]
```

External declaration of the system clock.

8.51.3.11 eventManager

```
EventManagerImpl eventManager
```

Global instance of the EventManager implementation.

Global instance of the EventManagerImpl class.

Definition at line 70 of file event_manager.cpp.

8.52 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;

static PowerEvent lastPowerState = PowerEvent::LOW_BATTERY;

static constexpr float FALL_RATE_THRESHOLD = -0.02f;

static constexpr int FALLING_TREND_REQUIRED = 3;

static constexpr float VOLTAGE_LOW_THRESHOLD = 4.7f;

static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f;

```

```
static int fallingTrendCount = 0;

bool lastSolarState = false;

bool lastUSBState = false;

extern DS3231 systemClock;

EventManagerImpl eventManager;


void EventManager::log_event(uint8_t group, uint8_t event) {
    mutex_enter_blocking(&eventMutex);

    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;
    if (eventCount < EVENT_BUFFER_SIZE) {
        eventCount++;
    }

    // Set persistence flag on buffer full or power events
    if (eventCount == EVENT_BUFFER_SIZE ||
        (group == static_cast<uint8_t>(EventGroup::POWER) &&
         event == static_cast<uint8_t>(PowerEvent::POWER_FALLING))) {
        needsPersistence = true;
        save_to_storage();
    }

    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];
}


void check_power_events(PowerManager& pm) {
    float currentVoltage = pm.get_voltage_5v();
    static float previousVoltage = 0.0f;
    float delta = currentVoltage - previousVoltage;
    previousVoltage = currentVoltage;

    if (delta < FALL_RATE_THRESHOLD) {
        fallingTrendCount++;
    } else {
        fallingTrendCount = 0;
    }

    if (fallingTrendCount >= FALLING_TREND_REQUIRED) {
        lastPowerState = PowerEvent::POWER_FALLING;
        EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_FALLING);
    }

    if (currentVoltage < PowerManager::VOLTAGE_LOW_THRESHOLD &&
        lastPowerState != PowerEvent::LOW_BATTERY) {
        lastPowerState = PowerEvent::LOW_BATTERY;
        EventEmitter::emit(EventGroup::POWER, PowerEvent::LOW_BATTERY);
    }
    else if (currentVoltage > PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&
             lastPowerState != PowerEvent::OVERCHARGE) {
        lastPowerState = PowerEvent::OVERCHARGE;
        EventEmitter::emit(EventGroup::POWER, PowerEvent::OVERCHARGE);
    }
    else if (currentVoltage >= PowerManager::VOLTAGE_LOW_THRESHOLD &&
             currentVoltage <= PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&
```

```
            lastPowerState != PowerEvent::POWER_NORMAL) {
        lastPowerState = PowerEvent::POWER_NORMAL;
        EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_NORMAL);
    }

    // Check solar charging state
    bool currentSolarState = pm.is_charging_solar();
    if (currentSolarState != lastSolarState) {
        if (currentSolarState) {
            EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_ACTIVE);
        } else {
            EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_INACTIVE);
        }
        lastSolarState = currentSolarState;
    }

    // Check USB connection state
    bool currentUSBState = pm.is_charging_usb();
    if (currentUSBState != lastUSBState) {
        if (currentUSBState) {
            EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_CONNECTED);
        } else {
            EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_DISCONNECTED);
        }
        lastUSBState = currentUSBState;
    }
}
```

8.53 lib/eventman/event_manager.h File Reference

```
#include "PowerManager.h"
#include <cstdint>
#include <string>
#include "pico/mutex.h"
#include "storage.h"
#include "utils.h"
```

Include dependency graph for event_manager.h:

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

Classes

- class EventLog

  Represents a single event log entry.
- class EventManager

  Manages the event logging system.
- class EventManagerImpl

  Implementation of the EventManager class.
- class EventEmitter

  Provides a static method for emitting events.

Macros

- #define EVENT_BUFFER_SIZE 10
- #define EVENT_LOG_FILE "/event_log.csv"

Enumerations

- enum class EventGroup : uint8_t {
  SYSTEM = 0x00 , POWER = 0x01 , COMMS = 0x02 , GPS = 0x03 ,
  CLOCK = 0x04 }
- enum class SystemEvent : uint8_t {
  BOOT = 0x01 , SHUTDOWN = 0x02 , WATCHDOG_RESET = 0x03 , CORE1_START = 0x04 ,
  CORE1_STOP = 0x05 }
- enum class PowerEvent : uint8_t {
  LOW_BATTERY = 0x01 , OVERCHARGE = 0x02 , POWER_FALLING = 0x03 , 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 }

Functions

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

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

  Checks power statuses and triggers events based on voltage trends.

Variables

- uint16_t id

  Sequence number.
- uint32_t timestamp

  Unix timestamp or system time.
- uint8_t group

  Event group identifier.
- uint8_t event

  Specific event identifier.
- class EventManager __attribute__
- EventManagerImpl eventManager

  Global instance of the EventManagerImpl class.

8.53.1 Macro Definition Documentation

8.53.1.1 EVENT_BUFFER_SIZE

```
#define EVENT_BUFFER_SIZE 10
```

Definition at line 11 of file event_manager.h.

8.53.1.2 EVENT_LOG_FILE

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

Definition at line 12 of file event_manager.h.

8.53.2 Enumeration Type Documentation

8.53.2.1 EventGroup

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

Enumerator

SYSTEM	
POWER	
COMMS	
GPS	
CLOCK	

Definition at line 15 of file event_manager.h.

8.53.2.2 SystemEvent

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

Enumerator

BOOT	
SHUTDOWN	
WATCHDOG_RESET	
CORE1_START	
CORE1_STOP	

Definition at line 24 of file event_manager.h.

8.53.2.3 PowerEvent

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

Enumerator

LOW_BATTERY	
OVERCHARGE	
POWER_FALLING	
POWER_NORMAL	
SOLAR_ACTIVE	
SOLAR_INACTIVE	
USB_CONNECTED	
USB_DISCONNECTED	

Definition at line 33 of file event_manager.h.

8.53.2.4 CommsEvent

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

Enumerator

RADIO_INIT	
RADIO_ERROR	
MSG_RECEIVED	
MSG_SENT	
UART_ERROR	

Definition at line 45 of file event_manager.h.

8.53.2.5 GPSEvent

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

Enumerator

LOCK	
LOST	
ERROR	
POWER_ON	
POWER_OFF	
DATA_READY	
PASS_THROUGH_START	
PASS_THROUGH_END	

Definition at line 54 of file event_manager.h.

8.53.2.6 ClockEvent

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

Enumerator

CHANGED	
GPS_SYNC	

Definition at line 66 of file event_manager.h.

8.53.3 Function Documentation

8.53.3.1 __attribute__()

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

8.53.3.2 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 10 of file event_manager.h.

8.53.3.3 check_power_events()

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

Checks power statuses and triggers events based on voltage trends.

Parameters

pm	Reference to the PowerManager object.
pm	Reference to the PowerManager object.

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

Definition at line 139 of file event_manager.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.53.4 Variable Documentation

8.53.4.1 id

```
uint16_t id
```

Sequence number.

Definition at line 1 of file event_manager.h.

8.53.4.2 timestamp

```
uint32_t timestamp
```

Unix timestamp or system time.

Definition at line 2 of file event_manager.h.

8.53.4.3 group

```
uint8_t group
```

Event group identifier.

Definition at line 3 of file event_manager.h.

8.53.4.4 event

```
uint8_t event
```

Specific event identifier.

Definition at line 4 of file event_manager.h.

8.53.4.5 __attribute__

```
class EventManager __attribute__
```

8.53.4.6 eventManager

```
EventManagerImpl eventManager [extern]
```

Global instance of the EventManagerImpl class.

Global instance of the EventManagerImpl class.

Definition at line 70 of file event_manager.cpp.

8.54 event_manager.h

Go to the documentation of this file.

```
#ifndef EVENT_MANAGER_H
#define EVENT_MANAGER_H

#include "PowerManager.h"
#include <cstdint>
#include <string>
#include "pico/mutex.h"
#include "storage.h"
#include "utils.h"

#define EVENT_BUFFER_SIZE 10
#define EVENT_LOG_FILE "/event_log.csv"

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

// System Events
enum class SystemEvent : uint8_t {
    BOOT           = 0x01,
    SHUTDOWN       = 0x02,
    WATCHDOG_RESET = 0x03,
    CORE1_START    = 0x04,
    CORE1_STOP     = 0x05
};

// Power Events
enum class PowerEvent : uint8_t {
    LOW_BATTERY      = 0x01,
    OVERCHARGE       = 0x02,
    POWER_FALLING    = 0x03,
    POWER_NORMAL     = 0x04,
    SOLAR_ACTIVE     = 0x05,
    SOLAR_INACTIVE   = 0x06,
    USB_CONNECTED    = 0x07,
```

```
    USB_DISCONNECTED = 0x08
};

// Communication Events
enum class CommsEvent : uint8_t {
    RADIO_INIT     = 0x01,
    RADIO_ERROR    = 0x02,
    MSG_RECEIVED   = 0x03,
    MSG_SENT       = 0x04,
    UART_ERROR     = 0x06
};

// GPS Events
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,
};

// Clock Events
enum class ClockEvent : uint8_t {
    CHANGED  = 0x01,
    GPS_SYNC = 0x02
};


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)
            , nextEventId(0)
            , needsPersistence(false)
        {
            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;
        volatile uint16_t nextEventId;
        bool needsPersistence;
    };


class EventManagerImpl : public EventManager {
    public:
```

```
        EventManagerImpl() {
            init(); // Safe to call virtual functions here
        }

        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) {
                for (size_t i = 0; i < eventCount; i++) {
                    fprintf(file, "%u;%lu;%u;%u\n",
                        events[i].id,
                        events[i].timestamp,
                        events[i].group,
                        events[i].event
                    );
                }
                fclose(file);
                needsPersistence = false;
                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)
        );
    }
};


void check_power_events(PowerManager& pm);

#endif
```

8.55 lib/location/gps_collector.cpp File Reference

```
#include "lib/location/gps_collector.h"
#include "utils.h"
#include "pico/time.h"
#include "lib/location/NMEA/nmea_data.h"
#include "event_manager.h"
#include <vector>
#include <ctime>
#include <cstring>
#include "DS3231.h"
#include <sstream>
```

Include dependency graph for gps_collector.cpp:

Macros

- #define MAX_RAW_DATA_LENGTH 1024

Functions

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

Variables

- NMEAData nmea_data

8.55.1 Macro Definition Documentation

8.55.1.1 MAX_RAW_DATA_LENGTH

```
#define MAX_RAW_DATA_LENGTH 1024
```

Definition at line 13 of file gps_collector.cpp.

8.55.2 Function Documentation

8.55.2.1 splitString()

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

Definition at line 17 of file gps_collector.cpp.

Here is the caller graph for this function:

8.55.2.2 collect_gps_data()

```
void collect_gps_data ()
```

Definition at line 27 of file gps_collector.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.55.3 Variable Documentation

8.55.3.1 nmea_data

```
NMEAData nmea_data  [extern]
```

Definition at line 3 of file NMEA_data.cpp.

8.56 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>

#define MAX_RAW_DATA_LENGTH 1024

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() {
    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;
            }
        }
    }
}
```

8.57 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 ()

8.57.1 Function Documentation

8.57.1.1 collect_gps_data()

```
void collect_gps_data ()
```

Definition at line 27 of file gps_collector.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.58 gps_collector.h

Go to the documentation of this file.

```
#ifndef GPS_COLLECTOR_H
#define GPS_COLLECTOR_H

#include <string>
#include "hardware/uart.h"
#include "lib/location/NMEA/nmea_data.h" // Include the new header
#include "pin_config.h"

// Function to collect GPS data from the UART
void collect_gps_data();

#endif
```

8.59 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

8.59.1 Variable Documentation

8.59.1.1 nmea_data

```
NMEAData nmea_data
```

Definition at line 3 of file NMEA_data.cpp.

8.60 NMEA_data.cpp

Go to the documentation of this file.

```
#include "lib/location/NMEA/NMEA_data.h"

NMEAData nmea_data; // Define the global instance

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;
}
```

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

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

Include dependency graph for NMEA_data.h:

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

Classes

- class NMEAData

Variables

- NMEAData nmea_data

8.61.1 Variable Documentation

8.61.1.1 nmea_data

```
NMEAData nmea_data  [extern]
```

Definition at line 3 of file NMEA_data.cpp.

8.62 NMEA_data.h

Go to the documentation of this file.

```
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/nmea_data.h
00002 #ifndef NMEA_DATA_H
00003 #define NMEA_DATA_H
00004
00005 #include <vector>
00006 #include <string>
00007 #include "pico/sync.h"
00008
00009 class NMEAData {
00010 public:
00011     NMEAData();
00012     void update_rmc_tokens(const std::vector<std::string>& tokens);
00013     void update_gga_tokens(const std::vector<std::string>& tokens);
00014
00015     std::vector<std::string> get_rmc_tokens() const;
```

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

private:
    std::vector<std::string> rmc_tokens_;
    std::vector<std::string> gga_tokens_;
    mutex_t rmc_mutex_;
    mutex_t gga_mutex_;
};

extern NMEAData nmea_data;

#endif
```

8.63 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

8.63.1 Variable Documentation

8.63.1.1 lora_cs_pin

```
const int lora_cs_pin = 17
```

Definition at line 4 of file pin_config.cpp.

8.63.1.2 lora_reset_pin

```
const int lora_reset_pin = 22
```

Definition at line 5 of file pin_config.cpp.

8.63.1.3 lora_irq_pin

```
const int lora_irq_pin = 28
```

Definition at line 6 of file pin_config.cpp.

8.63.1.4 lora_address_local

```
uint8_t lora_address_local = 37
```

Definition at line 8 of file pin_config.cpp.

8.63.1.5 lora_address_remote

```
uint8_t lora_address_remote = 21
```

Definition at line 9 of file pin_config.cpp.

8.64 pin_config.cpp

Go to the documentation of this file.

```
#include "pin_config.h"

// LoRa constants
const int lora_cs_pin = 17;          // LoRa radio chip select
const int lora_reset_pin = 22;       // LoRa radio reset
const int lora_irq_pin = 28;         // LoRa hardware interrupt pin

uint8_t lora_address_local = 37;     // address of this device
uint8_t lora_address_remote = 21;
```

8.65 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

8.65.1 Macro Definition Documentation

8.65.1.1 DEBUG_UART_PORT

```
#define DEBUG_UART_PORT uart0
```

Definition at line 8 of file pin_config.h.

8.65.1.2 DEBUG_UART_BAUD_RATE

```
#define DEBUG_UART_BAUD_RATE 115200
```

Definition at line 9 of file pin_config.h.

8.65.1.3 DEBUG_UART_TX_PIN

```
#define DEBUG_UART_TX_PIN 0
```

Definition at line 11 of file pin_config.h.

8.65.1.4 DEBUG_UART_RX_PIN

```
#define DEBUG_UART_RX_PIN 1
```

Definition at line 12 of file pin_config.h.

8.65.1.5 MAIN_I2C_PORT

```
#define MAIN_I2C_PORT i2c1
```

Definition at line 14 of file pin_config.h.

8.65.1.6 MAIN_I2C_SDA_PIN

```
#define MAIN_I2C_SDA_PIN 6
```

Definition at line 15 of file pin_config.h.

8.65.1.7 MAIN_I2C_SCL_PIN

```
#define MAIN_I2C_SCL_PIN 7
```

Definition at line 16 of file pin_config.h.

8.65.1.8 GPS_UART_PORT

```
#define GPS_UART_PORT uart1
```

Definition at line 19 of file pin_config.h.

8.65.1.9 GPS_UART_BAUD_RATE

```
#define GPS_UART_BAUD_RATE 9600
```

Definition at line 20 of file pin_config.h.

8.65.1.10 GPS_UART_TX_PIN

```
#define GPS_UART_TX_PIN 8
```

Definition at line 21 of file pin_config.h.

8.65.1.11 GPS_UART_RX_PIN

```
#define GPS_UART_RX_PIN 9
```

Definition at line 22 of file pin_config.h.

8.65.1.12 GPS_POWER_ENABLE_PIN

```
#define GPS_POWER_ENABLE_PIN 14
```

Definition at line 23 of file pin_config.h.

8.65.1.13 BUFFER_SIZE

```
#define BUFFER_SIZE 85
```

Definition at line 25 of file pin_config.h.

8.65.1.14 SD_SPI_PORT

```
#define SD_SPI_PORT spi1
```

Definition at line 28 of file pin_config.h.

8.65.1.15 SD_MISO_PIN

```
#define SD_MISO_PIN 12
```

Definition at line 29 of file pin_config.h.

8.65.1.16 SD_MOSI_PIN

```
#define SD_MOSI_PIN 11
```

Definition at line 30 of file pin_config.h.

8.65.1.17 SD_SCK_PIN

```
#define SD_SCK_PIN 10
```

Definition at line 31 of file pin_config.h.

8.65.1.18 SD_CS_PIN

```
#define SD_CS_PIN 13
```

Definition at line 32 of file pin_config.h.

8.65.1.19 SD_CARD_DETECT_PIN

```
#define SD_CARD_DETECT_PIN 28
```

Definition at line 33 of file pin_config.h.

8.65.1.20 SX1278_MISO

```
#define SX1278_MISO 16
```

Definition at line 35 of file pin_config.h.

8.65.1.21 SX1278_CS

```
#define SX1278_CS 17
```

Definition at line 36 of file pin_config.h.

8.65.1.22 SX1278_SCK

```
#define SX1278_SCK 18
```

Definition at line 37 of file pin_config.h.

8.65.1.23 SX1278_MOSI

```
#define SX1278_MOSI 19
```

Definition at line 38 of file pin_config.h.

8.65.1.24 SPI_PORT

```
#define SPI_PORT spi0
```

Definition at line 40 of file pin_config.h.

8.65.1.25 READ_BIT

```
#define READ_BIT 0x80
```

Definition at line 41 of file pin_config.h.

8.65.1.26 LORA_DEFAULT_SPI

```
#define LORA_DEFAULT_SPI spi0
```

Definition at line 43 of file pin_config.h.

8.65.1.27 LORA_DEFAULT_SPI_FREQUENCY

```
#define LORA_DEFAULT_SPI_FREQUENCY 8E6
```

Definition at line 44 of file pin_config.h.

8.65.1.28 LORA_DEFAULT_SS_PIN

```
#define LORA_DEFAULT_SS_PIN 17
```

Definition at line 45 of file pin_config.h.

8.65.1.29 LORA_DEFAULT_RESET_PIN

```
#define LORA_DEFAULT_RESET_PIN 22
```

Definition at line 46 of file pin_config.h.

8.65.1.30 LORA_DEFAULT_DIO0_PIN

```
#define LORA_DEFAULT_DIO0_PIN 20
```

Definition at line 47 of file pin_config.h.

8.65.1.31 PA_OUTPUT_RFO_PIN

```
#define PA_OUTPUT_RFO_PIN 11
```

Definition at line 49 of file pin_config.h.

8.65.1.32 PA_OUTPUT_PA_BOOST_PIN

```
#define PA_OUTPUT_PA_BOOST_PIN 12
```

Definition at line 50 of file pin_config.h.

8.65.2 Variable Documentation

8.65.2.1 lora_cs_pin

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

Definition at line 4 of file pin_config.cpp.

8.65.2.2 lora_reset_pin

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

Definition at line 5 of file pin_config.cpp.

8.65.2.3 lora_irq_pin

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

Definition at line 6 of file pin_config.cpp.

8.65.2.4 lora_address_local

```
uint8_t lora_address_local  [extern]
```

Definition at line 8 of file pin_config.cpp.

8.65.2.5 lora_address_remote

```
uint8_t lora_address_remote  [extern]
```

Definition at line 9 of file pin_config.cpp.

8.66 pin_config.h

Go to the documentation of this file.

```
00001 // pin_config.h
00002 #include <stdint.h>
00003
00004 #ifndef PIN_CONFIG_H
00005 #define PIN_CONFIG_H
00006
00007 //DEBUG uart
00008 #define DEBUG_UART_PORT uart0
00009 #define DEBUG_UART_BAUD_RATE 115200
00010
00011 #define DEBUG_UART_TX_PIN 0
00012 #define DEBUG_UART_RX_PIN 1
00013
00014 #define MAIN_I2C_PORT i2c1
00015 #define MAIN_I2C_SDA_PIN 6
00016 #define MAIN_I2C_SCL_PIN 7
00017
00018 // GPS configuration
00019 #define GPS_UART_PORT uart1
00020 #define GPS_UART_BAUD_RATE 9600
00021 #define GPS_UART_TX_PIN 8
00022 #define GPS_UART_RX_PIN 9
00023 #define GPS_POWER_ENABLE_PIN 14
00024
00025 #define BUFFER_SIZE 85  // NMEA sentences are usually under 85 chars
00026
00027 // SPI configuration for SD card
00028 #define SD_SPI_PORT spi1
00029 #define SD_MISO_PIN 12
00030 #define SD_MOSI_PIN 11
00031 #define SD_SCK_PIN 10
00032 #define SD_CS_PIN 13
00033 #define SD_CARD_DETECT_PIN 28
00034
00035 #define SX1278_MISO 16
00036 #define SX1278_CS   17
00037 #define SX1278_SCK  18
00038 #define SX1278_MOSI 19
00039
00040 #define SPI_PORT spi0
00041 #define READ_BIT 0x80
00042
00043 #define LORA_DEFAULT_SPI           spi0
00044 #define LORA_DEFAULT_SPI_FREQUENCY 8E6
00045 #define LORA_DEFAULT_SS_PIN        17
00046 #define LORA_DEFAULT_RESET_PIN     22
00047 #define LORA_DEFAULT_DIO0_PIN      20
00048
00049 #define PA_OUTPUT_RFO_PIN          11
00050 #define PA_OUTPUT_PA_BOOST_PIN     12
00051
00052
00053
00054 // LoRa constants - declare as extern
00055 extern const int lora_cs_pin;      // LoRa radio chip select
00056 extern const int lora_reset_pin;   // LoRa radio reset
00057 extern const int lora_irq_pin;     // LoRa hardware interrupt pin
00058 extern uint8_t lora_address_local; // address of this device
00059 extern uint8_t lora_address_remote;  // destination to send to
00060
00061
00062 #endif // PIN_CONFIG_H
```

8.67 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:

8.67.1 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.

8.68 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);
```

```

    _shuntRes[0] = 10;
    _shuntRes[1] = 10;
    _shuntRes[2] = 10;

    _filterRes[0] = 10;
    _filterRes[1] = 10;
    _filterRes[2] = 10;

    uint16_t manuf_id = get_manufacturer_id();
    uint16_t die_id = get_die_id();
    std::stringstream ss;
    ss « "INA3221 Manufacturer ID: 0x" « std::hex « manuf_id
              « ", Die ID: 0x" « die_id « std::endl;
    uart_print(ss.str(), VerbosityLevel::INFO);

    if (manuf_id == 0x5449 && die_id == 0x3220) {
       uart_print("INA3221 found and initialized.", VerbosityLevel::INFO);
        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;
```

```
00548     }
00549 }
```

8.69 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.

8.69.1 Detailed Description

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

Definition in file INA3221.h.

8.69.2 Enumeration Type Documentation

8.69.2.1 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.

8.69.2.2 ina3221_ch_t

```
enum ina3221_ch_t
```

Enumerator

INA3221_CH1	
INA3221_CH2	
INA3221_CH3	

Definition at line 23 of file INA3221.h.

8.69.2.3 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.

8.69.2.4 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.

8.69.2.5 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.

8.69.3 Variable Documentation

8.69.3.1 INA3221_CH_NUM

```
const int INA3221_CH_NUM = 3
```

Number of channels in INA3221.

Definition at line 30 of file INA3221.h.

8.69.3.2 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.

8.70 INA3221.h

Go to the documentation of this file.

```
00001 #ifndef BEASTDEVICES_INA3221_H
00002 #define BEASTDEVICES_INA3221_H
00003
00004 #include <stdint.h>
00005 #include <iostream>
00006 #include <hardware/i2c.h>
00007
00012 typedef enum {
00013     INA3221_ADDR40_GND = 0b1000000, // A0 pin -> GND
00014     INA3221_ADDR41_VCC = 0b1000001, // A0 pin -> VCC
00015     INA3221_ADDR42_SDA = 0b1000010, // A0 pin -> SDA
00016     INA3221_ADDR43_SCL = 0b1000011  // A0 pin -> SCL
00017 } ina3221_addr_t;
00018
00023 typedef enum {
00024     INA3221_CH1 = 0,
00025     INA3221_CH2,
00026     INA3221_CH3,
00027 } ina3221_ch_t;
00028
00030 const int INA3221_CH_NUM = 3;
00032 const int SHUNT_VOLTAGE_LSB_UV = 5;
00033
00034
00038 typedef enum {
00039     INA3221_REG_CONF = 0,
00040     INA3221_REG_CH1_SHUNTV,
00041     INA3221_REG_CH1_BUSV,
00042     INA3221_REG_CH2_SHUNTV,
00043     INA3221_REG_CH2_BUSV,
00044     INA3221_REG_CH3_SHUNTV,
00045     INA3221_REG_CH3_BUSV,
00046     INA3221_REG_CH1_CRIT_ALERT_LIM,
00047     INA3221_REG_CH1_WARNING_ALERT_LIM,
00048     INA3221_REG_CH2_CRIT_ALERT_LIM,
00049     INA3221_REG_CH2_WARNING_ALERT_LIM,
00050     INA3221_REG_CH3_CRIT_ALERT_LIM,
00051     INA3221_REG_CH3_WARNING_ALERT_LIM,
00052     INA3221_REG_SHUNTV_SUM,
00053     INA3221_REG_SHUNTV_SUM_LIM,
00054     INA3221_REG_MASK_ENABLE,
00055     INA3221_REG_PWR_VALID_HI_LIM,
00056     INA3221_REG_PWR_VALID_LO_LIM,
00057     INA3221_REG_MANUF_ID = 0xFE,
00058     INA3221_REG_DIE_ID = 0xFF
00059 } ina3221_reg_t;
00060
00065 typedef enum {
00066     INA3221_REG_CONF_CT_140US = 0,
00067     INA3221_REG_CONF_CT_204US,
00068     INA3221_REG_CONF_CT_332US,
00069     INA3221_REG_CONF_CT_588US,
00070     INA3221_REG_CONF_CT_1100US,
00071     INA3221_REG_CONF_CT_2116US,
00072     INA3221_REG_CONF_CT_4156US,
00073     INA3221_REG_CONF_CT_8244US
00074 } ina3221_conv_time_t;
00075
00080 typedef enum {
00081     INA3221_REG_CONF_AVG_1 = 0,
00082     INA3221_REG_CONF_AVG_4,
00083     INA3221_REG_CONF_AVG_16,
00084     INA3221_REG_CONF_AVG_64,
00085     INA3221_REG_CONF_AVG_128,
00086     INA3221_REG_CONF_AVG_256,
00087     INA3221_REG_CONF_AVG_512,
```

```
    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
```

8.71 lib/powerman/PowerManager.cpp File Reference

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

Include dependency graph for PowerManager.cpp:

8.72 PowerManager.cpp

Go to the documentation of this file.

```
#include "PowerManager.h"
#include <iostream>
#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::string MAN = "MAN " + std::to_string(ina3221_.get_manufacturer_id());
    std::string DIE = "DIE " + std::to_string(ina3221_.get_die_id());
    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;
}
```

8.73 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

8.74 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 = 4.7f;     // 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
```

8.75 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:

8.76 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);
}
```

8.77 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

8.77.1 Macro Definition Documentation

8.77.1.1 _BH1750_DEVICE_ID

```
#define _BH1750_DEVICE_ID 0xE1
```

Definition at line 7 of file BH1750.h.

8.77.1.2 _BH1750_MTREG_MIN

```
#define _BH1750_MTREG_MIN 31
```

Definition at line 8 of file BH1750.h.

8.77.1.3 _BH1750_MTREG_MAX

```
#define _BH1750_MTREG_MAX 254
```

Definition at line 9 of file BH1750.h.

8.77.1.4 _BH1750_DEFAULT_MTREG

```
#define _BH1750_DEFAULT_MTREG 69
```

Definition at line 10 of file BH1750.h.

8.78 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__
```

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

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

#include <iostream>

Include dependency graph for BH1750_WRAPPER.cpp:

8.80 BH1750_WRAPPER.cpp

Go to the documentation of this file.

```
// BH1750Wrapper.cpp
#include "BH1750_WRAPPER.h"
#include <string>
#include <iostream>

BH1750Wrapper::BH1750Wrapper() {
    sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE);
}

bool BH1750Wrapper::init() {
    initialized_ = sensor_.begin();
    return initialized_;
}

float BH1750Wrapper::read_data(SensorDataTypeIdentifier type) {
    if (type == SensorDataTypeIdentifier::LIGHT_LEVEL) {
        return sensor_.get_light_level();
    }
    return 0.0f;
}

bool BH1750Wrapper::is_initialized() const {
    return initialized_;
}

SensorType BH1750Wrapper::get_type() const {
    return SensorType::LIGHT;
}

bool BH1750Wrapper::configure(const std::map<std::string, std::string>& config) {
    for (const auto& [key, value] : config) {
        if (key == "measurement_mode") {
            if (value == "continuously_high_resolution") {
                sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE);
            }
            else if (value == "continuously_high_resolution_2") {
                sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE_2);
            }
            else if (value == "continuously_low_resolution") {
                sensor_.configure(BH1750::Mode::CONTINUOUS_LOW_RES_MODE);
            }
```

```
            else if (value == "one_time_high_resolution") {
                sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE);
            }
            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;
}
```

8.81 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

8.82 BH1750_WRAPPER.h

Go to the documentation of this file.

```
00001 #ifndef BH1750_WRAPPER_H
00002 #define BH1750_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "BH1750.h"
00006 #include <map>
00007 #include <string>
00008
00009 class BH1750Wrapper : public ISensor {
00010 private:
00011     BH1750 sensor_;
00012     bool initialized_ = false;
00013
00014 public:
00015     BH1750Wrapper();
00016     int get_i2c_addr();
00017     bool init() override;
00018     float read_data(SensorDataTypeIdentifier type) override;
00019     bool is_initialized() const override;
00020     SensorType get_type() const override;
00021
00022     bool configure(const std::map<std::string, std::string>& config);
00023
00024 };
00025
00026 #endif // BH1750_WRAPPER_H
```

8.83 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:

8.84 BME280.cpp

Go to the documentation of this file.

```
00001 // BME280.cpp
00002
00003 #include "BME280.h"
00004
00005 #include <iomanip>
00006 #include <vector>
00007 #include <algorithm>
00008 #include "hardware/i2c.h"
00009 #include "pico/binary_info.h"
00010 #include "pico/stdlib.h"
00011
00012 // BME280 (BME280) Class Implementation
00013
00014 BME280::BME280(i2c_inst_t* i2cPort, uint8_t address)
00015     : i2c_port(i2cPort), device_addr(address), calib_params{}, initialized_(false), t_fine(0) {
00016 }
00017
00018 bool BME280::init() {
00019     if (!i2c_port) {
00020         std::cerr « "Invalid I2C port.\n";
00021         return false;
00022     }
00023
00024     // Check device ID to confirm it's a BME280
00025     uint8_t reg = 0xD0; // Chip ID register
00026     uint8_t chip_id = 0;
00027     int ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
00028     if (ret != 1) {
00029         std::cerr « "Failed to write to BME280.\n";
00030         return false;
00031     }
00032     ret = i2c_read_blocking(i2c_port, device_addr, &chip_id, 1, false);
00033     if (ret != 1) {
00034         std::cerr « "Failed to read chip ID from BME280.\n";
00035         return false;
00036     }
00037     if (chip_id != 0x60) {
00038         std::cerr « "Device is not a BME280.\n";
00039         return false;
00040     }
00041
00042     // Configure sensor
00043     if (!configure_sensor()) {
00044         std::cerr « "Failed to configure BME280 sensor.\n";
00045         return false;
00046     }
00047
00048     // Retrieve calibration parameters
00049     if (!get_calibration_parameters()) {
00050         std::cerr « "Failed to retrieve calibration parameters.\n";
00051         return false;
00052     }
00053
```

```
    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;
}
```

8.85 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

8.86 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
```

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

`#include "BME280_WRAPPER.h"`

Include dependency graph for BME280_WRAPPER.cpp:

8.88 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;
}
```

8.89 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

8.90 BME280_WRAPPER.h

Go to the documentation of this file.

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

```
#define BME280_WRAPPER_H

#include "ISensor.h"
#include "BME280.h"

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;

};

#endif // BME280_WRAPPER_H
```

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

`#include "HMC5883L.h"`

Include dependency graph for HMC5883L.cpp:

8.92 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;
}
```

8.93 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

8.94 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
```

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

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

`#include <cstdio>`

Include dependency graph for HMC5883L_WRAPPER.cpp:

8.96 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;
}
```

8.97 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

8.98 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;

private:
    HMC5883L sensor_;
    bool initialized_;
};

#endif
```

8.99 lib/sensors/ISensor.cpp File Reference

Implements the SensorWrapper class for managing different sensor types.

```
#include "ISensor.h"
#include "lib/sensors/BH1750/BH1750_WRAPPER.h"
#include "lib/sensors/BME280/BME280_WRAPPER.h"
#include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
```

Include dependency graph for ISensor.cpp:

8.99.1 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.

8.100 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"



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;
}
```

8.101 lib/sensors/ISensor.h File Reference

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

```
#include <string>
```

Include dependency graph for ISensor.h:

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

Classes

- class ISensor
- class SensorWrapper

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

Enumerations

- enum class SensorType { LIGHT , ENVIRONMENT , MAGNETOMETER , IMU }
- enum class 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 }

8.101.1 Enumeration Type Documentation

8.101.1.1 SensorType

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

Enumerator

LIGHT	
ENVIRONMENT	
MAGNETOMETER	
IMU	

Definition at line 10 of file ISensor.h.

8.101.1.2 SensorDataTypeIdentifier

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

Enumerator

LIGHT_LEVEL	
TEMPERATURE	
PRESSURE	
HUMIDITY	
MAG_FIELD_X	
MAG_FIELD_Y	
MAG_FIELD_Z	
GYRO_X	
GYRO_Y	
GYRO_Z	
ACCEL_X	
ACCEL_Y	
ACCEL_Z	

Definition at line 17 of file ISensor.h.

8.102 ISensor.h

Go to the documentation of this file.

```
00001 // ISensor.h
00002 #ifndef ISENSOR_H
00003 #define ISENSOR_H
00004
00005 #include <cstdint>
00006 #include "hardware/i2c.h"
00007 #include <map>
00008 #include <string>
00009
00010 enum class SensorType {
00011     LIGHT,          // BH1750
00012     ENVIRONMENT,    // BME280
00013     MAGNETOMETER,   // HMC5883L
00014     IMU             // MPU6050
00015 };
00016
00017 enum class SensorDataTypeIdentifier {
00018     LIGHT_LEVEL,
00019     TEMPERATURE,
00020     PRESSURE,
00021     HUMIDITY,
00022     MAG_FIELD_X,
00023     MAG_FIELD_Y,
```

```
    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;
};

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);

private:
    std::map<SensorType, ISensor*> sensors;
    SensorWrapper();
};

#endif // ISENSOR_H
```

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

8.104 MPU6050.cpp

Go to the documentation of this file.

```
```

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

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

8.106 MPU6050.h

Go to the documentation of this file.

```
00001
```

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

8.108 MPU6050_WRAPPER.cpp

Go to the documentation of this file.

```
00001
```

8.109 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

8.110 MPU6050_WRAPPER.h

Go to the documentation of this file.

```
00001 #ifndef BH1750_WRAPPER_H
00002 #define BH1750_WRAPPER_H
00003
```

```
#include "ISensor.h"
#include "MPU6050.h"
#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
```

8.111 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

8.111.1 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.

8.111.2 Function Documentation

8.111.2.1 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:

8.111.3 Variable Documentation

8.111.3.1 sd_card_mounted

```
bool sd_card_mounted = false
```

Definition at line 17 of file storage.cpp.

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

8.113 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

8.113.1 Function Documentation

8.113.1.1 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:

8.113.1.2 fs_open_file()

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

8.113.1.3 fs_write_file()

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

8.113.1.4 fs_read_file()

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

8.113.1.5 fs_close_file()

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

8.113.1.6 fs_file_exists()

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

8.113.2 Variable Documentation

8.113.2.1 sd_card_mounted

```
bool sd_card_mounted  [extern]
```

Definition at line 17 of file storage.cpp.

8.114 storage.h

Go to the documentation of this file.

```
00001 #ifndef STORAGE_H
00002 #define STORAGE_H
00003
00004 #include <stdio.h>
00005 #include <string.h>
00006 #include <hardware/clocks.h>
00007 #include <hardware/flash.h>
00008 #include "blockdevice/flash.h"
00009 #include "blockdevice/sd.h"
00010 #include "filesystem/littlefs.h"
00011 #include "filesystem/vfs.h"
00012 #include "pin_config.h"
00013 #include "lfs.h"
00014 #include "filesystem/fat.h"
00015
00016
00017 extern bool sd_card_mounted;
00018
00019 struct FileHandle {
00020     int fd;
00021     bool is_open;
00022 };
00023
00024 bool fs_init(void);
00025 FileHandle fs_open_file(const char* filename, const char* mode);
00026 ssize_t fs_write_file(FileHandle& handle, const void* buffer, size_t size);
00027 ssize_t fs_read_file(FileHandle& handle, void* buffer, size_t size);
00028 bool fs_close_file(FileHandle& handle);
00029 bool fs_file_exists(const char* filename);
00030
00031 #endif
```

```

// 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);
// }
```

8.115 lib/utils.cpp File Reference

Implementation of utility functions for the Kubisat firmware.

```
#include "utils.h"
#include "pico/multicore.h"
#include "pico/sync.h"
#include <vector>
#include <queue>
```

Include dependency graph for utils.cpp:

Functions

- std::string get_level_color (VerbosityLevel level)

  Gets ANSI color code for verbosity level.

- std::string get_level_prefix (VerbosityLevel level)

  Gets text prefix for verbosity level.
- void uart_print (const std::string &msg, VerbosityLevel level, bool logToFile, uart_inst_t *uart)

  Prints a message to the UART with a timestamp and core number.
- uint16_t crc16 (const uint8_t *data, size_t length)

  Calculates the CRC16 checksum of the given data.

Variables

- static mutex_t uart_mutex

  Mutex for UART access protection.
- VerbosityLevel g_uart_verbosity = VerbosityLevel::EVENT

  Global verbosity level setting.

8.115.1 Detailed Description

Implementation of utility functions for the Kubisat firmware.

Definition in file utils.cpp.

8.115.2 Function Documentation

8.115.2.1 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 26 of file utils.cpp.

Here is the caller graph for this function:

8.115.2.2 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 43 of file utils.cpp.

Here is the caller graph for this function:

8.115.2.3 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 62 of file utils.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.115.2.4 crc16()

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

Calculates the CRC16 checksum of the given data.

Calculates CRC16 checksum.

Parameters

data	A pointer to the data buffer.
length	The length of the data in bytes.

Returns

The CRC16 checksum.

Calculates the CRC16 checksum using the standard algorithm.

Definition at line 97 of file utils.cpp.

8.115.3 Variable Documentation

8.115.3.1 uart_mutex

```
mutex_t uart_mutex  [static]
```

Mutex for UART access protection.

Definition at line 14 of file utils.cpp.

8.115.3.2 g_uart_verbosity

```
VerbosityLevel g_uart_verbosity = VerbosityLevel::EVENT
```

Global verbosity level setting.

Global verbosity level setting for UART output.

Definition at line 18 of file utils.cpp.

8.116 utils.cpp

Go to the documentation of this file.

```
00001 #include "utils.h"
00002 #include "pico/multicore.h"
00003 #include "pico/sync.h"
00004 #include <vector>
00005 #include <queue>
00006
00011
00012
00014 static mutex_t uart_mutex;
00015
00016
00018 VerbosityLevel g_uart_verbosity = VerbosityLevel::EVENT;
00019
00020
00026 std::string get_level_color(VerbosityLevel level) {
00027     switch (level) {
00028         case VerbosityLevel::ERROR:   return ANSI_RED;
00029         case VerbosityLevel::WARNING: return ANSI_YELLOW;
00030         case VerbosityLevel::INFO:    return ANSI_GREEN;
00031         case VerbosityLevel::DEBUG:   return ANSI_BLUE;
```

```
        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();

    // Create formatted message with color
    std::string color = get_level_color(level);
    std::string prefix = get_level_prefix(level);
    std::string msgToSend = "[" + std::to_string(timestamp) + "ms] - Core " +
                            std::to_string(core_num) + ": " +
                            color + prefix + ANSI_RESET + msg + "\r\n";

    // Print to UART
    mutex_enter_blocking(&uart_mutex);
    uart_puts(uart, msgToSend.c_str());
    mutex_exit(&uart_mutex);
}


uint16_t crc16(const uint8_t *data, size_t length) {
    uint16_t crc = 0xFFFF;
    for (size_t i = 0; i < length; i++) {
        crc ^= data[i];
        for (int j = 0; j < 8; j++) {
            if (crc & 0x0001) {
                crc = (crc » 1) ^ 0xA001;
            } else {
                crc »= 1;
            }
        }
    }
    return crc;
}
```

8.117 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.
- uint16_t crc16 (const uint8_t *data, size_t length)

  Calculates CRC16 checksum.

Variables

- VerbosityLevel g_uart_verbosity

  Global verbosity level setting for UART output.

8.117.1 Detailed Description

Utility functions and definitions for the Kubisat firmware.

Contains UART logging, color definitions, and CRC calculations

Definition in file utils.h.

8.117.2 Macro Definition Documentation

8.117.2.1 ANSI_RED

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

ANSI escape codes for terminal color output.

Definition at line 20 of file utils.h.

8.117.2.2 ANSI_GREEN

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

Definition at line 21 of file utils.h.

8.117.2.3 ANSI_YELLOW

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

Definition at line 22 of file utils.h.

8.117.2.4 ANSI_BLUE

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

Definition at line 23 of file utils.h.

8.117.2.5 ANSI_CYAN

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

Definition at line 24 of file utils.h.

8.117.2.6 ANSI_RESET

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

Definition at line 25 of file utils.h.

8.117.3 Enumeration Type Documentation

8.117.3.1 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.

8.117.4 Function Documentation

8.117.4.1 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 62 of file utils.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.117.4.2 crc16()

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

Calculates CRC16 checksum.

Parameters

data	Pointer to data buffer
length	Length of data in bytes

Returns

Calculated CRC16 value

Calculates CRC16 checksum.

Parameters

data	A pointer to the data buffer.
length	The length of the data in bytes.

Returns

The CRC16 checksum.

Calculates the CRC16 checksum using the standard algorithm.

Definition at line 97 of file utils.cpp.

8.117.5 Variable Documentation

8.117.5.1 g_uart_verbosity

```
VerbosityLevel g_uart_verbosity [extern]
```

Global verbosity level setting for UART output.

Controls which messages are displayed:

- SILENT (0): No output
- ERROR (1): Only errors
- WARNING (2): Warnings and errors
- INFO (3): Normal operation information
- DEBUG (4): Detailed debug information

Note

Can be changed at runtime through the command interface

See also

VerbosityLevel
handle_verbosity

Global verbosity level setting for UART output.

Definition at line 18 of file utils.cpp.

8.118 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);



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

#endif
```

8.119 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_systems ()
- int main ()

Variables

- PowerManager powerManager (MAIN_I2C_PORT)
- DS3231 systemClock (MAIN_I2C_PORT)
- char buffer [BUFFER_SIZE]
- int bufferIndex = 0

8.119.1 Macro Definition Documentation

8.119.1.1 LOG_FILENAME

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

Definition at line 3 of file main.cpp.

8.119.2 Function Documentation

8.119.2.1 core1_entry()

```
void core1_entry ()
```

Definition at line 11 of file main.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.119.2.2 init_systems()

```
bool init_systems ()
```

Definition at line 20 of file main.cpp.

Here is the call graph for this function:

Here is the caller graph for this function:

8.119.2.3 main()

```
int main (
            void )
```

Definition at line 92 of file main.cpp.

Here is the call graph for this function:

8.119.3 Variable Documentation

8.119.3.1 powerManager

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

8.119.3.2 systemClock

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

8.119.3.3 buffer

```
char buffer[BUFFER_SIZE]
```

Definition at line 8 of file main.cpp.

8.119.3.4 bufferIndex

```
int bufferIndex = 0
```

Definition at line 9 of file main.cpp.

8.120 main.cpp

Go to the documentation of this file.

```
#include "includes.h"

#define LOG_FILENAME "/log.txt"

PowerManager powerManager(MAIN_I2C_PORT);
DS3231 systemClock(MAIN_I2C_PORT);

char buffer[BUFFER_SIZE];
int bufferIndex = 0;

void core1_entry() {
    uart_print("Starting core 1", VerbosityLevel::DEBUG);
    while (true) {
        collect_gps_data();
        check_power_events(powerManager);
        sleep_ms(100);
    }
}

bool init_systems() {
    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));

    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);

    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));

    if (true)
    {
        gpio_init(GPS_POWER_ENABLE_PIN);
        gpio_set_dir(GPS_POWER_ENABLE_PIN, GPIO_OUT);
        gpio_put(GPS_POWER_ENABLE_PIN, 1);
    }
    system("color");

    uart_print("System init started.");

    bool radioInitSuccess = false;

    radioInitSuccess = initialize_radio();

    bool sdInitDone = fs_init();
    if (sdInitDone) {
        FILE *fp = fopen(LOG_FILENAME, "w");
        if (fp) {
            uart_print("Log file opened.");
            int bytesWritten = fprintf(fp, "System init started.\n");
            uart_print("Written " + std::to_string(bytesWritten) + " bytes.");
            int closeStatus = fclose(fp);
            uart_print("Close file status: " + std::to_string(closeStatus));

            // Get file size
            struct stat file_stat;
            if (stat(LOG_FILENAME, &file_stat) == 0) {
                size_t fileSize = file_stat.st_size;
                uart_print("File size: " + std::to_string(fileSize) + " bytes");
            } else {
                uart_print("Failed to get file size");
            }

            // Print file path (assuming it's in the root directory)
            uart_print("File path: /" + std::string(LOG_FILENAME));
        } else {
            uart_print("Failed to open log file for writing.");
        }
    }

    uart_print("SD card init: " + std::to_string(sdInitDone));
    std::string bootString = "System init completed @ " +
std::to_string(to_ms_since_boot(get_absolute_time())) + " ms";
```

```
    uart_print(bootString);

    Frame boot = frame_build(ExecutionResult::INFO, 0, 0, "HELLO");
    send_frame(boot);

    uart_print("System init done.");
    return radioInitSuccess;
}


int main()
{
    init_systems();
    multicore_launch_core1(core1_entry);

    gpio_put(PICO_DEFAULT_LED_PIN, 0);
    bool powerManagerInitStatus = powerManager.initialize();
    if (powerManagerInitStatus)
    {
        std::map<std::string, std::string> powerConfig = {
            {"operating_mode", "continuous"},
            {"averaging_mode", "16"},
        };
        powerManager.configure(powerConfig);
    } else {
        uart_print("Power manager init error");
    }
    gpio_put(PICO_DEFAULT_LED_PIN, 1);


    for (int i = 5; i > 0; --i)
    {
        std::string intro = "Main loop starts in " + std::to_string(i) + " seconds...";
        uart_print(intro);
        gpio_put(PICO_DEFAULT_LED_PIN, (i%2==0));
        sleep_ms(1000);
    }
    gpio_put(PICO_DEFAULT_LED_PIN, 1);

    Frame boot = frame_build(ExecutionResult::INFO, 0, 0, "START");
    send_frame(boot);

    while (true)
    {
        int packetSize = LoRa.parse_packet();
        if (packetSize)
        {
            on_receive(packetSize);
        }

        handle_uart_input();
    }

    return 0;
}

// BH1750     0X23
// INA3221    0X40
// BME280     0X76
// DS3231     0X86
```

8.121 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 ()

8.121.1 Function Documentation

8.121.1.1 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:

8.121.1.2 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:

8.121.1.3 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:

8.121.1.4 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:

8.122 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]);
}
```

8.123 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(ExecutionResult::SUCCESS, 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(ExecutionResult::ERROR, 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(ExecutionResult::INFO, 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() {
00005     Frame frame = create_test_frame();
00006     std::string encoded = frame_encode(frame);
00007
00008     TEST_ASSERT_NOT_EQUAL(0, encoded.length());
00009     TEST_ASSERT_TRUE(encoded.find(FRAME_BEGIN) != std::string::npos);
00010     TEST_ASSERT_TRUE(encoded.find(FRAME_END) != std::string::npos);
00011     TEST_ASSERT_TRUE(encoded.find("test_value") != std::string::npos);
00012 }
00013
00014 void test_frame_decode_basic() {
00015     Frame original = create_test_frame();
00016     std::string encoded = frame_encode(original);
00017     Frame decoded = frame_decode(encoded);
00018
00019     TEST_ASSERT_EQUAL(original.direction, decoded.direction);
00020     TEST_ASSERT_EQUAL(original.group, decoded.group);
00021     TEST_ASSERT_EQUAL(original.command, decoded.command);
00022     TEST_ASSERT_EQUAL_STRING(original.value.c_str(), decoded.value.c_str());
00023     TEST_ASSERT_EQUAL_STRING(original.unit.c_str(), decoded.unit.c_str());
00024 }
00025
00026 void test_frame_decode_invalid_header() {
00027     std::string invalid_frame = "INVALID" + std::string(1, DELIMITER) + "rest_of_frame";
00028     bool exceptionThrown = false;
00029
00030     try {
00031         Frame decoded = frame_decode(invalid_frame);
00032     } catch (const std::runtime_error& e) {
00033         exceptionThrown = true;
00034     } catch (...) {
00035         // Catch any other exceptions to avoid crashing the test
00036     }
00037
00038     TEST_ASSERT_TRUE(exceptionThrown);
00039 }
```

8.127 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/test_runner.cpp File Reference

```
#include "unity.h"
#include "pico/stdlib.h"
#include "hardware/uart.h"
```

Include dependency graph for test_runner.cpp:

Functions

- void test_frame_encode_basic (void)
- void test_frame_decode_basic (void)
- void test_frame_decode_invalid_header (void)
- void test_frame_build_success (void)
- void test_frame_build_error (void)
- void test_frame_build_info (void)
- void test_operation_type_conversion (void)
- void test_value_unit_type_conversion (void)
- void test_exception_type_conversion (void)
- void test_hex_string_conversion (void)
- int main (void)

8.129.1 Function Documentation

8.129.1.1 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:

8.129.1.2 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:

8.129.1.3 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:

8.129.1.4 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:

8.129.1.5 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:

8.129.1.6 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:

8.129.1.7 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:

8.129.1.8 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:

8.129.1.9 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:

8.129.1.10 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:

8.129.1.11 main()

```
int main (
            void )
```

Definition at line 18 of file test_runner.cpp.

Here is the call graph for this function:

8.130 test_runner.cpp

Go to the documentation of this file.

```
00001 // test_runner.cpp
00002 #include "unity.h"
00003 #include "pico/stdlib.h"
00004 #include "hardware/uart.h"
00005
00006 // External test function declarations
00007 extern void test_frame_encode_basic(void);
00008 extern void test_frame_decode_basic(void);
00009 extern void test_frame_decode_invalid_header(void);
00010 extern void test_frame_build_success(void);
00011 extern void test_frame_build_error(void);
00012 extern void test_frame_build_info(void);
00013 extern void test_operation_type_conversion(void);
00014 extern void test_value_unit_type_conversion(void);
00015 extern void test_exception_type_conversion(void);
00016 extern void test_hex_string_conversion(void);
00017
00018 int main(void) {
00019     stdio_init_all();
00020     uart_init(uart0, 115200);
```

```
    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
    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
    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
    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);
    uart_puts(uart0, "end converter tests\n");

    return UNITY_END();
}
```

Index

_BH1750_DEFAULT_MTREG
BH1750.h, 332
_BH1750_DEVICE_ID
BH1750.h, 331
_BH1750_MTREG_MAX
BH1750.h, 331
_BH1750_MTREG_MIN
BH1750.h, 331
__attribute__
event_manager.h, 293, 295
__empty
LoRa-RP2040.h, 252
_dio0
LoRaClass, 141
_filterRes
INA3221, 112
_frequency
LoRaClass, 141
_i2c
INA3221, 112
_i2c_addr
BH1750, 58
INA3221, 112
_implicitHeaderMode
LoRaClass, 142
_masken_reg
INA3221, 112
_onCadDone
LoRaClass, 142
_onReceive
LoRaClass, 142
_onTxDone
LoRaClass, 142
_packetIndex
LoRaClass, 141
_read
INA3221, 108
_reset
LoRaClass, 141
_shuntRes
INA3221, 112
_spi
LoRaClass, 141
_ss
LoRaClass, 141
_write
INA3221, 109
~EventManager
EventManager, 91
~ISensor
ISensor, 113

ACCEL_X
ISensor.h, 352
ACCEL_Y
ISensor.h, 352
ACCEL_Z
ISensor.h, 352
ADDR_SDO_HIGH
BME280, 64
ADDR_SDO_LOW
BME280, 64
address
HMC5883L, 103
Alert Functions, 48
enable_alerts, 50
get_crit_alert, 51
get_power_valid_alert, 53
get_warn_alert, 51
set_alert_latch, 53
set_crit_alert_limit, 49
set_power_valid_limit, 50
set_warn_alert_limit, 49
ANS
protocol.h, 263
ANSI_BLUE
utils.h, 370
ANSI_CYAN
utils.h, 370
ANSI_GREEN
utils.h, 370
ANSI_RED
utils.h, 370
ANSI_RESET
utils.h, 370
ANSI_YELLOW
utils.h, 370
available
LoRaClass, 121
availableForWrite
Print, 162
avg_mode
INA3221::conf_reg_t, 77

bcd_to_bin
DS3231, 84
begin
BH1750, 56
Configuration Functions, 38

LoRaClass, 117
beginPacket
LoRaClass, 118
BH1750, 55
_i2c_addr, 58
begin, 56
BH1750, 56
configure, 56
CONTINUOUS_HIGH_RES_MODE, 56
CONTINUOUS_HIGH_RES_MODE_2, 56
CONTINUOUS_LOW_RES_MODE, 56
get_light_level, 57
Mode, 55
ONE_TIME_HIGH_RES_MODE, 56
ONE_TIME_HIGH_RES_MODE_2, 56
ONE_TIME_LOW_RES_MODE, 56
POWER_ON, 56
RESET, 56
UNCONFIGURED_POWER_DOWN, 56
write8, 57
BH1750.h
_BH1750_DEFAULT_MTREG, 332
_BH1750_DEVICE_ID, 331
_BH1750_MTREG_MAX, 331
_BH1750_MTREG_MIN, 331
BH1750Wrapper, 58
BH1750Wrapper, 59
configure, 60
get_i2c_addr, 59
get_type, 60
init, 59
initialized_, 60
is_initialized, 60
read_data, 59
sensor_, 60
BIN
Print.h, 260
bin_to_bcd
DS3231, 83
BME280, 61
ADDR_SDO_HIGH, 64
ADDR_SDO_LOW, 64
BME280, 62
calib_params, 65
configure_sensor, 64
convert_humidity, 63
convert_pressure, 63
convert_temperature, 63
device_addr, 65
get_calibration_parameters, 64
i2c_port, 65
init, 63
initialized_, 65
NUM_CALIB_PARAMS, 70
read_raw_all, 63
REG_CONFIG, 65
REG_CTRL_HUM, 65
REG_CTRL_MEAS, 65
REG_DIG_H1, 69
REG_DIG_H2, 69
REG_DIG_H3, 69
REG_DIG_H4, 69
REG_DIG_H5, 70
REG_DIG_H6, 70
REG_DIG_P1_LSB, 67
REG_DIG_P1_MSB, 67
REG_DIG_P2_LSB, 67
REG_DIG_P2_MSB, 67
REG_DIG_P3_LSB, 67
REG_DIG_P3_MSB, 67
REG_DIG_P4_LSB, 68
REG_DIG_P4_MSB, 68
REG_DIG_P5_LSB, 68
REG_DIG_P5_MSB, 68
REG_DIG_P6_LSB, 68
REG_DIG_P6_MSB, 68
REG_DIG_P7_LSB, 68
REG_DIG_P7_MSB, 68
REG_DIG_P8_LSB, 69
REG_DIG_P8_MSB, 69
REG_DIG_P9_LSB, 69
REG_DIG_P9_MSB, 69
REG_DIG_T1_LSB, 66
REG_DIG_T1_MSB, 66
REG_DIG_T2_LSB, 66
REG_DIG_T2_MSB, 66
REG_DIG_T3_LSB, 67
REG_DIG_T3_MSB, 67
REG_HUMIDITY_MSB, 66
REG_PRESSURE_MSB, 66
REG_RESET, 66
REG_TEMPERATURE_MSB, 66
reset, 63
t_fine, 65
BME280CalibParam, 70
dig_h1, 72
dig_h2, 72
dig_h3, 72
dig_h4, 73
dig_h5, 73
dig_h6, 73
dig_p1, 71
dig_p2, 71
dig_p3, 71
dig_p4, 71
dig_p5, 72
dig_p6, 72
dig_p7, 72
dig_p8, 72
dig_p9, 72
dig_t1, 71
dig_t2, 71
dig_t3, 71
BME280Wrapper, 73
BME280Wrapper, 74
configure, 75

get_type, 75
init, 75
initialized_, 75
is_initialized, 75
read_data, 75
sensor_, 75
BOOL
protocol.h, 263
BOOT
event_manager.h, 291
buffer
main.cpp, 377
BUFFER_SIZE
pin_config.h, 309
bufferIndex
main.cpp, 377
BUILD_NUMBER
build_number.h, 181
build_number.h, 181
BUILD_NUMBER, 181
bus_conv_time
INA3221::conf_reg_t, 77

calculate_checksum
storage_commands_utils.cpp, 209
storage_commands_utils.h, 212
calib_params
BME280, 65
century
ds3231_data_t, 86
ch1_en
INA3221::conf_reg_t, 77
ch2_en
INA3221::conf_reg_t, 77
ch3_en
INA3221::conf_reg_t, 77
CHANGED
event_manager.h, 293
channelActivityDetection
LoRaClass, 124
charging_solar_active_
PowerManager, 155
charging_usb_active_
PowerManager, 156
check_power_alerts
PowerManager, 154
check_power_events
event_manager.cpp, 285
event_manager.h, 293
clearWriteError
Print, 160
CLOCK
event_manager.h, 291
Clock Commands, 1
Clock Management Commands, 11
handle_clock_sync_interval, 13
handle_get_last_sync_time, 13
handle_time, 11
handle_timezone_offset, 12
systemClock, 14
clock_commands.cpp
CLOCK_GROUP, 191
CLOCK_SYNC_INTERVAL, 192
LAST_SYNC_TIME, 192
TIME, 191
TIMEZONE_OFFSET, 191
clock_enable
DS3231, 81
CLOCK_GROUP
clock_commands.cpp, 191
clock_mutex_
DS3231, 85
CLOCK_SYNC_INTERVAL
clock_commands.cpp, 192
ClockEvent
event_manager.h, 292
collect_gps_data
gps_collector.cpp, 298
gps_collector.h, 301
command
Frame, 100
Command System, 14
CommandHandler, 15
commandHandlers, 16
CommandMap, 15
execute_command, 15
CommandAccessLevel
protocol.h, 263
CommandHandler
Command System, 15
frame.cpp, 228
commandHandlers
Command System, 16
CommandMap
Command System, 15
COMMS
event_manager.h, 291
CommsEvent
event_manager.h, 292
communication.cpp
initialize_radio, 213
interval, 215
lastPrintTime, 215
lastReceiveTime, 215
lastSendTime, 214
msgCount, 214
outgoing, 214
communication.h
determine_unit, 226
frame_build, 224
frame_decode, 223
frame_encode, 222
frame_process, 221
handle_uart_input, 218
initialize_radio, 217
on_receive, 217
send_frame, 219

send_frame_lora, 220
send_frame_uart, 219
send_message, 219
split_and_send_message, 221
Configuration Functions, 36
begin, 38
get_die_id, 39
get_manufacturer_id, 39
INA3221, 37
read_register, 40
reset, 38
set_averaging_mode, 44
set_bus_conversion_time, 45
set_bus_measurement_disable, 44
set_bus_measurement_enable, 43
set_mode_continuous, 41
set_mode_power_down, 41
set_mode_triggered, 42
set_shunt_conversion_time, 45
set_shunt_measurement_disable, 43
set_shunt_measurement_enable, 42
configure
BH1750, 56
BH1750Wrapper, 60
BME280Wrapper, 75
HMC5883LWrapper, 106
ISensor, 113
MPU6050Wrapper, 147
PowerManager, 153
configure_sensor
BME280, 64
CONTINUOUS_HIGH_RES_MODE
BH1750, 56
CONTINUOUS_HIGH_RES_MODE_2
BH1750, 56
CONTINUOUS_LOW_RES_MODE
BH1750, 56
conv_ready
INA3221::masken_reg_t, 143
convert_humidity
BME280, 63
convert_pressure
BME280, 63
convert_temperature
BME280, 63
core1_entry
main.cpp, 375
CORE1_START
event_manager.h, 291
CORE1_STOP
event_manager.h, 291
crc
LoRaClass, 131
crc16
utils.cpp, 366
utils.h, 372
create_test_frame
test_frame_common.h, 389
crit_alert_ch1
INA3221::masken_reg_t, 144
crit_alert_ch2
INA3221::masken_reg_t, 144
crit_alert_ch3
INA3221::masken_reg_t, 144
crit_alert_latch_en
INA3221::masken_reg_t, 144

DATA_COMMAND
storage_commands.cpp, 206
DATA_READY
event_manager.h, 292
date
ds3231_data_t, 86
DATETIME
protocol.h, 263
day
ds3231_data_t, 86
days_of_week
DS3231.h, 189
DEBUG
utils.h, 371
DEBUG_UART_BAUD_RATE
pin_config.h, 308
DEBUG_UART_PORT
pin_config.h, 308
DEBUG_UART_RX_PIN
pin_config.h, 308
DEBUG_UART_TX_PIN
pin_config.h, 308
DEC
Print.h, 259
DELIMITER
protocol.h, 269
determine_unit
communication.h, 226
device_addr
BME280, 65
Diagnostic Commands, 17
handle_enter_bootloader_mode, 20
handle_get_build_version, 18
handle_get_commands_list, 17
handle_verbosity, 19
dig_h1
BME280CalibParam, 72
dig_h2
BME280CalibParam, 72
dig_h3
BME280CalibParam, 72
dig_h4
BME280CalibParam, 73
dig_h5
BME280CalibParam, 73
dig_h6
BME280CalibParam, 73
dig_p1
BME280CalibParam, 71
dig_p2

BME280CalibParam, 71
dig_p3
BME280CalibParam, 71
dig_p4
BME280CalibParam, 71
dig_p5
BME280CalibParam, 72
dig_p6
BME280CalibParam, 72
dig_p7
BME280CalibParam, 72
dig_p8
BME280CalibParam, 72
dig_p9
BME280CalibParam, 72
dig_t1
BME280CalibParam, 71
dig_t2
BME280CalibParam, 71
dig_t3
BME280CalibParam, 71
direction
Frame, 100
disableCrc
LoRaClass, 129
disableInvertIQ
LoRaClass, 130
DS3231, 78
bcd_to_bin, 84
bin_to_bcd, 83
clock_enable, 81
clock_mutex_, 85
DS3231, 79
ds3231_addr, 85
get_time, 79
get_unix_time, 81
i2c, 85
i2c_read_reg, 82
i2c_write_reg, 82
read_temperature, 80
set_time, 79
set_unix_time, 80
DS3231.h
days_of_week, 189
DS3231_CONTROL_REG, 189
DS3231_CONTROL_STATUS_REG, 189
DS3231_DATE_REG, 188
DS3231_DAY_REG, 188
DS3231_DEVICE_ADRESS, 188
DS3231_HOURS_REG, 188
DS3231_MINUTES_REG, 188
DS3231_MONTH_REG, 188
DS3231_SECONDS_REG, 188
DS3231_TEMPERATURE_LSB_REG, 189
DS3231_TEMPERATURE_MSB_REG, 189
DS3231_YEAR_REG, 189
FRIDAY, 189
MONDAY, 189
SATURDAY, 189
SUNDAY, 189
THURSDAY, 189
TUESDAY, 189
WEDNESDAY, 189
ds3231_addr
DS3231, 85
DS3231_CONTROL_REG
DS3231.h, 189
DS3231_CONTROL_STATUS_REG
DS3231.h, 189
ds3231_data_t, 85
century, 86
date, 86
day, 86
hours, 86
minutes, 86
month, 86
seconds, 86
year, 86
DS3231_DATE_REG
DS3231.h, 188
DS3231_DAY_REG
DS3231.h, 188
DS3231_DEVICE_ADRESS
DS3231.h, 188
DS3231_HOURS_REG
DS3231.h, 188
DS3231_MINUTES_REG
DS3231.h, 188
DS3231_MONTH_REG
DS3231.h, 188
DS3231_SECONDS_REG
DS3231.h, 188
DS3231_TEMPERATURE_LSB_REG
DS3231.h, 189
DS3231_TEMPERATURE_MSB_REG
DS3231.h, 189
DS3231_YEAR_REG
DS3231.h, 189
dumpRegisters
LoRaClass, 132

emit
EventEmitter, 87
enable_alerts
Alert Functions, 50
enableCrc
LoRaClass, 128
enableInvertIQ
LoRaClass, 129
end
LoRaClass, 117
END_COMMAND
storage_commands.cpp, 206
endPacket
LoRaClass, 118
ENVIRONMENT
ISensor.h, 352

ERR
protocol.h, 263
ERROR
event_manager.h, 292
protocol.h, 263
utils.h, 371
EVENT
utils.h, 371
event
event_manager.h, 294
EventLog, 90
Event Commands, 21
handle_get_event_count, 22
handle_get_last_events, 21
EVENT_BUFFER_SIZE
event_manager.h, 291
EVENT_LOG_FILE
event_manager.h, 291
event_manager.cpp
check_power_events, 285
eventLogId, 285
eventManager, 287
FALL_RATE_THRESHOLD, 286
FALLING_TREND_REQUIRED, 286
fallingTrendCount, 286
lastPowerState, 285
lastSolarState, 286
lastUSBState, 287
systemClock, 287
VOLTAGE_LOW_THRESHOLD, 286
VOLTAGE_OVERCHARGE_THRESHOLD, 286
event_manager.h
__attribute__, 293, 295
BOOT, 291
CHANGED, 293
check_power_events, 293
CLOCK, 291
ClockEvent, 292
COMMS, 291
CommsEvent, 292
CORE1_START, 291
CORE1_STOP, 291
DATA_READY, 292
ERROR, 292
event, 294
EVENT_BUFFER_SIZE, 291
EVENT_LOG_FILE, 291
EventGroup, 291
eventManager, 295
GPS, 291
GPS_SYNC, 293
GPSEvent, 292
group, 294
id, 294
LOCK, 292
LOST, 292
LOW_BATTERY, 292
MSG_RECEIVED, 292
MSG_SENT, 292
OVERCHARGE, 292
PASS_THROUGH_END, 292
PASS_THROUGH_START, 292
POWER, 291
POWER_FALLING, 292
POWER_NORMAL, 292
POWER_OFF, 292
POWER_ON, 292
PowerEvent, 291
RADIO_ERROR, 292
RADIO_INIT, 292
SHUTDOWN, 291
SOLAR_ACTIVE, 292
SOLAR_INACTIVE, 292
SYSTEM, 291
SystemEvent, 291
timestamp, 294
to_string, 293
UART_ERROR, 292
USB_CONNECTED, 292
USB_DISCONNECTED, 292
WATCHDOG_RESET, 291
eventCount
EventManager, 95
EventEmitter, 87
emit, 87
EventGroup
event_manager.h, 291
EventLog, 88
event, 90
group, 89
id, 89
timestamp, 89
to_string, 89
eventLogId
event_manager.cpp, 285
EventManager, 90
~EventManager, 91
eventCount, 95
EventManager, 91
eventMutex, 95
events, 95
get_event, 93
get_event_count, 93
init, 92
load_from_storage, 94
log_event, 92
needsPersistence, 95
nextEventId, 95
save_to_storage, 94
writeIndex, 95
eventManager
event_manager.cpp, 287
event_manager.h, 295
EventManagerImpl, 96
EventManagerImpl, 97
load_from_storage, 98

save_to_storage, 98
eventMutex
EventManager, 95
eventRegister
frame.cpp, 232
events
EventManager, 95
exception_type_to_string
protocol.h, 264
utils_converters.cpp, 279
ExceptionType
protocol.h, 263
execute_command
Command System, 15
ExecutionResult
protocol.h, 262
explicitHeaderMode
LoRaClass, 132

FALL_RATE_THRESHOLD
event_manager.cpp, 286
PowerManager, 155
FALLING_TREND_REQUIRED
event_manager.cpp, 286
PowerManager, 155
fallingTrendCount
event_manager.cpp, 286
fd
FileHandle, 99
FileHandle, 99
fd, 99
is_open, 99
flush
LoRaClass, 122
Print, 175
footer
Frame, 101
Frame, 99
command, 100
direction, 100
footer, 101
group, 100
header, 100
operationType, 100
unit, 100
value, 100
frame.cpp
CommandHandler, 228
eventRegister, 232
frame_build, 231
frame_decode, 229
frame_encode, 228
frame_process, 230
FRAME_BEGIN
protocol.h, 269
frame_build
communication.h, 224
frame.cpp, 231
frame_decode
communication.h, 223
frame.cpp, 229
frame_encode
communication.h, 222
frame.cpp, 228
FRAME_END
protocol.h, 269
frame_process
communication.h, 221
frame.cpp, 230
FRIDAY
DS3231.h, 189
fs_close_file
storage.h, 361
fs_file_exists
storage.h, 361
fs_init
storage.cpp, 357
storage.h, 359
fs_open_file
storage.h, 360
fs_read_file
storage.h, 360
fs_write_file
storage.h, 360

g_uart_verbosity
utils.cpp, 367
utils.h, 373
GET
protocol.h, 263
get_calibration_parameters
BME280, 64
get_crit_alert
Alert Functions, 51
get_current
INA3221, 111
get_current_charge_solar
PowerManager, 151
get_current_charge_total
PowerManager, 152
get_current_charge_usb
PowerManager, 152
get_current_draw
PowerManager, 152
get_current_ma
Measurement Functions, 47
get_die_id
Configuration Functions, 39
get_event
EventManager, 93
get_event_count
EventManager, 93
get_gga_tokens
NMEAData, 149
get_i2c_addr
BH1750Wrapper, 59
get_instance
SensorWrapper, 177

get_level_color
utils.cpp, 363
get_level_prefix
utils.cpp, 364
get_light_level
BH1750, 57
get_manufacturer_id
Configuration Functions, 39
get_power_valid_alert
Alert Functions, 53
get_rmc_tokens
NMEAData, 149
get_shunt_voltage
Measurement Functions, 46
get_time
DS3231, 79
get_type
BH1750Wrapper, 60
BME280Wrapper, 75
HMC5883LWrapper, 105
ISensor, 113
MPU6050Wrapper, 147
get_unix_time
DS3231, 81
get_voltage
Measurement Functions, 48
get_voltage_5v
PowerManager, 152
get_voltage_battery
PowerManager, 152
get_warn_alert
Alert Functions, 51
getSignalBandwidth
LoRaClass, 135
getSpreadingFactor
LoRaClass, 135
getWriteError
Print, 160
gga_mutex_
NMEAData, 149
gga_tokens_
NMEAData, 149
GPS
event_manager.h, 291
GPS Commands, 23
handle_enable_gps_uart_passthrough, 24
handle_get_gga_data, 26
handle_get_rmc_data, 25
handle_gps_power_status, 23
gps_collector.cpp
collect_gps_data, 298
MAX_RAW_DATA_LENGTH, 298
nmea_data, 299
splitString, 298
gps_collector.h
collect_gps_data, 301
GPS_POWER_ENABLE_PIN
pin_config.h, 309
GPS_SYNC
event_manager.h, 293
GPS_UART_BAUD_RATE
pin_config.h, 309
GPS_UART_PORT
pin_config.h, 309
GPS_UART_RX_PIN
pin_config.h, 309
GPS_UART_TX_PIN
pin_config.h, 309
GPSEvent
event_manager.h, 292
group
event_manager.h, 294
EventLog, 89
Frame, 100
GYRO_X
ISensor.h, 352
GYRO_Y
ISensor.h, 352
GYRO_Z
ISensor.h, 352

handle_clock_sync_interval
Clock Management Commands, 13
handle_enable_gps_uart_passthrough
GPS Commands, 24
handle_enter_bootloader_mode
Diagnostic Commands, 20
handle_file_download
Storage Commands, 33
handle_get_build_version
Diagnostic Commands, 18
handle_get_commands_list
Diagnostic Commands, 17
handle_get_current_charge_solar
Power Commands, 30
handle_get_current_charge_total
Power Commands, 31
handle_get_current_charge_usb
Power Commands, 29
handle_get_current_draw
Power Commands, 32
handle_get_event_count
Event Commands, 22
handle_get_gga_data
GPS Commands, 26
handle_get_last_events
Event Commands, 21
handle_get_last_sync_time
Clock Management Commands, 13
handle_get_power_manager_ids
Power Commands, 27
handle_get_rmc_data
GPS Commands, 25
handle_get_voltage_5v
Power Commands, 28
handle_get_voltage_battery
Power Commands, 28

handle_gps_power_status
GPS Commands, 23
handle_list_files
Storage Commands, 34
handle_mount
Storage Commands, 35
handle_time
Clock Management Commands, 11
handle_timezone_offset
Clock Management Commands, 12
handle_uart_input
communication.h, 218
receive.cpp, 271
handle_verbosity
Diagnostic Commands, 19
handleDio0Rise
LoRaClass, 134
header
Frame, 100
HEX
Print.h, 259
hex_string_to_bytes
protocol.h, 266
utils_converters.cpp, 282
HMC5883L, 101
address, 103
HMC5883L, 101
i2c, 103
init, 102
read, 102
read_register, 103
write_register, 102
HMC5883LWrapper, 104
configure, 106
get_type, 105
HMC5883LWrapper, 105
init, 105
initialized_, 106
is_initialized, 105
read_data, 105
sensor_, 106
hours
ds3231_data_t, 86
HUMIDITY
ISensor.h, 352

i2c
DS3231, 85
HMC5883L, 103
i2c_port
BME280, 65
i2c_read_reg
DS3231, 82
i2c_write_reg
DS3231, 82
id
event_manager.h, 294
EventLog, 89
idle
LoRaClass, 124
implicitHeaderMode
LoRaClass, 133
IMU
ISensor.h, 352
INA3221, 106
_filterRes, 112
_i2c, 112
_i2c_addr, 112
_masken_reg, 112
_read, 108
_shuntRes, 112
_write, 109
Configuration Functions, 37
get_current, 111
INA3221 Power Monitor, 36
INA3221.h
INA3221_ADDR40_GND, 321
INA3221_ADDR41_VCC, 321
INA3221_ADDR42_SDA, 321
INA3221_ADDR43_SCL, 321
ina3221_addr_t, 320
ina3221_avg_mode_t, 322
INA3221_CH1, 321
INA3221_CH2, 321
INA3221_CH3, 321
INA3221_CH_NUM, 322
ina3221_ch_t, 321
ina3221_conv_time_t, 321
INA3221_REG_CH1_BUSV, 321
INA3221_REG_CH1_CRIT_ALERT_LIM, 321
INA3221_REG_CH1_SHUNTV, 321
INA3221_REG_CH1_WARNING_ALERT_LIM, 321
INA3221_REG_CH2_BUSV, 321
INA3221_REG_CH2_CRIT_ALERT_LIM, 321
INA3221_REG_CH2_SHUNTV, 321
INA3221_REG_CH2_WARNING_ALERT_LIM, 321
INA3221_REG_CH3_BUSV, 321
INA3221_REG_CH3_CRIT_ALERT_LIM, 321
INA3221_REG_CH3_SHUNTV, 321
INA3221_REG_CH3_WARNING_ALERT_LIM, 321
INA3221_REG_CONF, 321
INA3221_REG_CONF_AVG_1, 322
INA3221_REG_CONF_AVG_1024, 322
INA3221_REG_CONF_AVG_128, 322
INA3221_REG_CONF_AVG_16, 322
INA3221_REG_CONF_AVG_256, 322
INA3221_REG_CONF_AVG_4, 322
INA3221_REG_CONF_AVG_512, 322
INA3221_REG_CONF_AVG_64, 322
INA3221_REG_CONF_CT_1100US, 322
INA3221_REG_CONF_CT_140US, 322
INA3221_REG_CONF_CT_204US, 322
INA3221_REG_CONF_CT_2116US, 322
INA3221_REG_CONF_CT_332US, 322

INA3221_REG_CONF_CT_4156US, 322
INA3221_REG_CONF_CT_588US, 322
INA3221_REG_CONF_CT_8244US, 322
INA3221_REG_DIE_ID, 321
INA3221_REG_MANUF_ID, 321
INA3221_REG_MASK_ENABLE, 321
INA3221_REG_PWR_VALID_HI_LIM, 321
INA3221_REG_PWR_VALID_LO_LIM, 321
INA3221_REG_SHUNTV_SUM, 321
INA3221_REG_SHUNTV_SUM_LIM, 321
ina3221_reg_t, 321
SHUNT_VOLTAGE_LSB_UV, 322
INA3221::conf_reg_t, 76
avg_mode, 77
bus_conv_time, 77
ch1_en, 77
ch2_en, 77
ch3_en, 77
mode_bus_en, 76
mode_continious_en, 76
mode_shunt_en, 76
reset, 77
shunt_conv_time, 77
INA3221::masken_reg_t, 142
conv_ready, 143
crit_alert_ch1, 144
crit_alert_ch2, 144
crit_alert_ch3, 144
crit_alert_latch_en, 144
pwr_valid_alert, 143
reserved, 145
shunt_sum_alert, 144
shunt_sum_en_ch1, 145
shunt_sum_en_ch2, 145
shunt_sum_en_ch3, 144
timing_ctrl_alert, 143
warn_alert_ch1, 144
warn_alert_ch2, 143
warn_alert_ch3, 143
warn_alert_latch_en, 144
ina3221_
PowerManager, 155
INA3221_ADDR40_GND
INA3221.h, 321
INA3221_ADDR41_VCC
INA3221.h, 321
INA3221_ADDR42_SDA
INA3221.h, 321
INA3221_ADDR43_SCL
INA3221.h, 321
ina3221_addr_t
INA3221.h, 320
ina3221_avg_mode_t
INA3221.h, 322
INA3221_CH1
INA3221.h, 321
INA3221_CH2
INA3221.h, 321
INA3221_CH3
INA3221.h, 321
INA3221_CH_NUM
INA3221.h, 322
ina3221_ch_t
INA3221.h, 321
ina3221_conv_time_t
INA3221.h, 321
INA3221_REG_CH1_BUSV
INA3221.h, 321
INA3221_REG_CH1_CRIT_ALERT_LIM
INA3221.h, 321
INA3221_REG_CH1_SHUNTV
INA3221.h, 321
INA3221_REG_CH1_WARNING_ALERT_LIM
INA3221.h, 321
INA3221_REG_CH2_BUSV
INA3221.h, 321
INA3221_REG_CH2_CRIT_ALERT_LIM
INA3221.h, 321
INA3221_REG_CH2_SHUNTV
INA3221.h, 321
INA3221_REG_CH2_WARNING_ALERT_LIM
INA3221.h, 321
INA3221_REG_CH3_BUSV
INA3221.h, 321
INA3221_REG_CH3_CRIT_ALERT_LIM
INA3221.h, 321
INA3221_REG_CH3_SHUNTV
INA3221.h, 321
INA3221_REG_CH3_WARNING_ALERT_LIM
INA3221.h, 321
INA3221_REG_CONF
INA3221.h, 321
INA3221_REG_CONF_AVG_1
INA3221.h, 322
INA3221_REG_CONF_AVG_1024
INA3221.h, 322
INA3221_REG_CONF_AVG_128
INA3221.h, 322
INA3221_REG_CONF_AVG_16
INA3221.h, 322
INA3221_REG_CONF_AVG_256
INA3221.h, 322
INA3221_REG_CONF_AVG_4
INA3221.h, 322
INA3221_REG_CONF_AVG_512
INA3221.h, 322
INA3221_REG_CONF_AVG_64
INA3221.h, 322
INA3221_REG_CONF_CT_1100US
INA3221.h, 322
INA3221_REG_CONF_CT_140US
INA3221.h, 322
INA3221_REG_CONF_CT_204US
INA3221.h, 322
INA3221_REG_CONF_CT_2116US
INA3221.h, 322

INA3221_REG_CONF_CT_332US
INA3221.h, 322
INA3221_REG_CONF_CT_4156US
INA3221.h, 322
INA3221_REG_CONF_CT_588US
INA3221.h, 322
INA3221_REG_CONF_CT_8244US
INA3221.h, 322
INA3221_REG_DIE_ID
INA3221.h, 321
INA3221_REG_MANUF_ID
INA3221.h, 321
INA3221_REG_MASK_ENABLE
INA3221.h, 321
INA3221_REG_PWR_VALID_HI_LIM
INA3221.h, 321
INA3221_REG_PWR_VALID_LO_LIM
INA3221.h, 321
INA3221_REG_SHUNTV_SUM
INA3221.h, 321
INA3221_REG_SHUNTV_SUM_LIM
INA3221.h, 321
ina3221_reg_t
INA3221.h, 321
includes.h, 182
INF
protocol.h, 263
INFO
protocol.h, 263
utils.h, 371
init
BH1750Wrapper, 59
BME280, 63
BME280Wrapper, 75
EventManager, 92
HMC5883L, 102
HMC5883LWrapper, 105
ISensor, 113
MPU6050Wrapper, 147
init_systems
main.cpp, 375
initialize
PowerManager, 151
initialize_radio
communication.cpp, 213
communication.h, 217
initialized_
BH1750Wrapper, 60
BME280, 65
BME280Wrapper, 75
HMC5883LWrapper, 106
MPU6050Wrapper, 147
PowerManager, 155
Interface
protocol.h, 264
interval
communication.cpp, 215
INVALID_OPERATION
protocol.h, 264
INVALID_PARAM
protocol.h, 264
IRQ_CAD_DETECTED_MASK
LoRa-RP2040.cpp, 236
IRQ_CAD_DONE_MASK
LoRa-RP2040.cpp, 237
IRQ_PAYLOAD_CRC_ERROR_MASK
LoRa-RP2040.cpp, 241
IRQ_RX_DONE_MASK
LoRa-RP2040.cpp, 241
IRQ_TX_DONE_MASK
LoRa-RP2040.cpp, 241
is_charging_solar
PowerManager, 153
is_charging_usb
PowerManager, 153
is_initialized
BH1750Wrapper, 60
BME280Wrapper, 75
HMC5883LWrapper, 105
ISensor, 113
MPU6050Wrapper, 147
is_open
FileHandle, 99
ISensor, 112
~ISensor, 113
configure, 113
get_type, 113
init, 113
is_initialized, 113
read_data, 113
ISensor.h
ACCEL_X, 352
ACCEL_Y, 352
ACCEL_Z, 352
ENVIRONMENT, 352
GYRO_X, 352
GYRO_Y, 352
GYRO_Z, 352
HUMIDITY, 352
IMU, 352
LIGHT, 352
LIGHT_LEVEL, 352
MAG_FIELD_X, 352
MAG_FIELD_Y, 352
MAG_FIELD_Z, 352
MAGNETOMETER, 352
PRESSURE, 352
SensorDataTypeIdentifier, 352
SensorType, 351
TEMPERATURE, 352
ISR_PREFIX
LoRa-RP2040.cpp, 242
isTransmitting
LoRaClass, 134

LAST_SYNC_TIME
clock_commands.cpp, 192

lastPowerState
event_manager.cpp, 285
lastPrintTime
communication.cpp, 215
lastReceiveTime
communication.cpp, 215
lastSendTime
communication.cpp, 214
lastSolarState
event_manager.cpp, 286
lastUSBState
event_manager.cpp, 287
lib/clock/DS3231.cpp, 183
lib/clock/DS3231.h, 187, 190
lib/comms/commands/clock_commands.cpp, 190, 192
lib/comms/commands/commands.cpp, 194
lib/comms/commands/commands.h, 195, 197
lib/comms/commands/diagnostic_commands.cpp, 198
lib/comms/commands/event_commands.cpp, 199, 200
lib/comms/commands/gps_commands.cpp, 201
lib/comms/commands/power_commands.cpp, 203, 204
lib/comms/commands/storage_commands.cpp, 205, 207
lib/comms/commands/storage_commands_utils.cpp, 209, 211
lib/comms/commands/storage_commands_utils.h, 211, 213
lib/comms/communication.cpp, 213, 215
lib/comms/communication.h, 216, 227
lib/comms/frame.cpp, 227, 233
lib/comms/LoRa/LoRa-RP2040.cpp, 234, 243
lib/comms/LoRa/LoRa-RP2040.h, 251, 253
lib/comms/LoRa/Print.cpp, 254
lib/comms/LoRa/Print.h, 259, 260
lib/comms/protocol.h, 261, 269
lib/comms/receive.cpp, 270, 272
lib/comms/send.cpp, 273, 277
lib/comms/utils_converters.cpp, 278, 283
lib/eventman/event_manager.cpp, 283, 287
lib/eventman/event_manager.h, 289, 295
lib/location/gps_collector.cpp, 297, 299
lib/location/gps_collector.h, 300, 302
lib/location/NMEA/NMEA_data.cpp, 302, 303
lib/location/NMEA/NMEA_data.h, 303, 304
lib/pin_config.cpp, 305, 306
lib/pin_config.h, 307, 313
lib/powerman/INA3221/INA3221.cpp, 314
lib/powerman/INA3221/INA3221.h, 319, 323
lib/powerman/PowerManager.cpp, 325
lib/powerman/PowerManager.h, 327, 328
lib/sensors/BH1750/BH1750.cpp, 329
lib/sensors/BH1750/BH1750.h, 330, 332
lib/sensors/BH1750/BH1750_WRAPPER.cpp, 332, 333
lib/sensors/BH1750/BH1750_WRAPPER.h, 334, 335
lib/sensors/BME280/BME280.cpp, 335, 336
lib/sensors/BME280/BME280.h, 339, 340
lib/sensors/BME280/BME280_WRAPPER.cpp, 342
lib/sensors/BME280/BME280_WRAPPER.h, 343
lib/sensors/HMC5883L/HMC5883L.cpp, 344
lib/sensors/HMC5883L/HMC5883L.h, 345, 346
lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp, 346, 347
lib/sensors/HMC5883L/HMC5883L_WRAPPER.h, 348, 349
lib/sensors/ISensor.cpp, 349, 350
lib/sensors/ISensor.h, 350, 352
lib/sensors/MPU6050/MPU6050.cpp, 353
lib/sensors/MPU6050/MPU6050.h, 354
lib/sensors/MPU6050/MPU6050_WRAPPER.cpp, 354
lib/sensors/MPU6050/MPU6050_WRAPPER.h, 354, 355
lib/storage/storage.cpp, 356, 358
lib/storage/storage.h, 358, 361
lib/utils.cpp, 362, 367
lib/utils.h, 368, 374
LIGHT
ISensor.h, 352
LIGHT_LEVEL
ISensor.h, 352
LIST_FILES_COMMAND
storage_commands.cpp, 207
load_from_storage
EventManager, 94
EventManagerImpl, 98
LOCK
event_manager.h, 292
log_event
EventManager, 92
LOG_FILENAME
main.cpp, 375
LORA
protocol.h, 264
LoRa
LoRa-RP2040.cpp, 242
LoRa-RP2040.h, 252
LoRa-RP2040.cpp
IRQ_CAD_DETECTED_MASK, 236
IRQ_CAD_DONE_MASK, 237
IRQ_PAYLOAD_CRC_ERROR_MASK, 241
IRQ_RX_DONE_MASK, 241
IRQ_TX_DONE_MASK, 241
ISR_PREFIX, 242
LoRa, 242
MAX_PKT_LENGTH, 242
MODE_CAD, 240
MODE_LONG_RANGE_MODE, 240
MODE_RX_CONTINUOUS, 241
MODE_RX_SINGLE, 241
MODE_SLEEP, 240
MODE_STDBY, 241
MODE_TX, 241
PA_BOOST, 241
REG_DETECTION_OPTIMIZE, 239
REG_DETECTION_THRESHOLD, 239
REG_DIO_MAPPING_1, 240
REG_FIFO, 236

REG_FIFO_ADDR_PTR, 237
REG_FIFO_RX_BASE_ADDR, 237
REG_FIFO_RX_CURRENT_ADDR, 237
REG_FIFO_TX_BASE_ADDR, 237
REG_FREQ_ERROR_LSB, 239
REG_FREQ_ERROR_MID, 239
REG_FREQ_ERROR_MSB, 239
REG_FRF_LSB, 236
REG_FRF_MID, 236
REG_FRF_MSB, 236
REG_INVERTIQ, 239
REG_INVERTIQ2, 240
REG_IRQ_FLAGS, 237
REG_LNA, 237
REG_MODEM_CONFIG_1, 238
REG_MODEM_CONFIG_2, 238
REG_MODEM_CONFIG_3, 239
REG_OCP, 236
REG_OP_MODE, 236
REG_PA_CONFIG, 236
REG_PA_DAC, 240
REG_PAYLOAD_LENGTH, 238
REG_PKT_RSSI_VALUE, 238
REG_PKT_SNR_VALUE, 238
REG_PREAMBLE_LSB, 238
REG_PREAMBLE_MSB, 238
REG_RSSI_VALUE, 238
REG_RSSI_WIDEBAND, 239
REG_RX_NB_BYTES, 237
REG_SYNC_WORD, 240
REG_VERSION, 240
RF_MID_BAND_THRESHOLD, 242
RSSI_OFFSET_HF_PORT, 242
RSSI_OFFSET_LF_PORT, 242
LoRa-RP2040.h
__empty, 252
LoRa, 252
lora_address_local
pin_config.cpp, 306
pin_config.h, 312
lora_address_remote
pin_config.cpp, 306
pin_config.h, 313
lora_cs_pin
pin_config.cpp, 305
pin_config.h, 312
LORA_DEFAULT_DIO0_PIN
pin_config.h, 312
LORA_DEFAULT_RESET_PIN
pin_config.h, 311
LORA_DEFAULT_SPI
pin_config.h, 311
LORA_DEFAULT_SPI_FREQUENCY
pin_config.h, 311
LORA_DEFAULT_SS_PIN
pin_config.h, 311
lora_irq_pin
pin_config.cpp, 306
pin_config.h, 312
lora_reset_pin
pin_config.cpp, 305
pin_config.h, 312
LoRaClass, 114
_dio0, 141
_frequency, 141
_implicitHeaderMode, 142
_onCadDone, 142
_onReceive, 142
_onTxDone, 142
_packetIndex, 141
_reset, 141
_spi, 141
_ss, 141
available, 121
begin, 117
beginPacket, 118
channelActivityDetection, 124
crc, 131
disableCrc, 129
disableInvertIQ, 130
dumpRegisters, 132
enableCrc, 128
enableInvertIQ, 129
end, 117
endPacket, 118
explicitHeaderMode, 132
flush, 122
getSignalBandwidth, 135
getSpreadingFactor, 135
handleDio0Rise, 134
idle, 124
implicitHeaderMode, 133
isTransmitting, 134
LoRaClass, 117
noCrc, 131
onCadDone, 123
onDio0Rise, 140
onReceive, 123
onTxDone, 123
packetFrequencyError, 119
packetRssi, 119
packetSnr, 119
parse_packet, 118
peek, 122
random, 131
read, 122
readRegister, 136
receive, 124
rssi, 120
set_pins, 132
setCodingRate4, 127
setFrequency, 126
setGain, 130
setLdoFlag, 136
setOCP, 130
setPreambleLength, 128

setSignalBandwidth, 127
setSPI, 132
setSPIFrequency, 132
setSpreadingFactor, 127
setSyncWord, 128
setTxPower, 126
singleTransfer, 139
sleep, 125
write, 120, 121
writeRegister, 137
LOST
event_manager.h, 292
LOW_BATTERY
event_manager.h, 292

MAG_FIELD_X
ISensor.h, 352
MAG_FIELD_Y
ISensor.h, 352
MAG_FIELD_Z
ISensor.h, 352
MAGNETOMETER
ISensor.h, 352
main
main.cpp, 376
test_runner.cpp, 397
main.cpp, 374
buffer, 377
bufferIndex, 377
core1_entry, 375
init_systems, 375
LOG_FILENAME, 375
main, 376
powerManager, 377
systemClock, 377
MAIN_I2C_PORT
pin_config.h, 308
MAIN_I2C_SCL_PIN
pin_config.h, 309
MAIN_I2C_SDA_PIN
pin_config.h, 309
MAX_BLOCK_SIZE
storage_commands.cpp, 206
MAX_PKT_LENGTH
LoRa-RP2040.cpp, 242
MAX_RAW_DATA_LENGTH
gps_collector.cpp, 298
Measurement Functions, 46
get_current_ma, 47
get_shunt_voltage, 46
get_voltage, 48
MILIAMP
protocol.h, 263
minutes
ds3231_data_t, 86
Mode
BH1750, 55
mode_bus_en
INA3221::conf_reg_t, 76
MODE_CAD
LoRa-RP2040.cpp, 240
mode_continious_en
INA3221::conf_reg_t, 76
MODE_LONG_RANGE_MODE
LoRa-RP2040.cpp, 240
MODE_RX_CONTINUOUS
LoRa-RP2040.cpp, 241
MODE_RX_SINGLE
LoRa-RP2040.cpp, 241
mode_shunt_en
INA3221::conf_reg_t, 76
MODE_SLEEP
LoRa-RP2040.cpp, 240
MODE_STDBY
LoRa-RP2040.cpp, 241
MODE_TX
LoRa-RP2040.cpp, 241
MONDAY
DS3231.h, 189
month
ds3231_data_t, 86
MOUNT_COMMAND
storage_commands.cpp, 207
MPU6050Wrapper, 145
configure, 147
get_type, 147
init, 147
initialized_, 147
is_initialized, 147
MPU6050Wrapper, 146
read_data, 147
sensor_, 147
MSG_RECEIVED
event_manager.h, 292
MSG_SENT
event_manager.h, 292
msgCount
communication.cpp, 214

needsPersistence
EventManager, 95
nextEventId
EventManager, 95
nmea_data
gps_collector.cpp, 299
NMEA_data.cpp, 302
NMEA_data.h, 304
NMEA_data.cpp
nmea_data, 302
NMEA_data.h
nmea_data, 304
NMEAData, 148
get_gga_tokens, 149
get_rmc_tokens, 149
gga_mutex_, 149
gga_tokens_, 149
NMEAData, 148
rmc_mutex_, 149

rmc_tokens_, 149
update_gga_tokens, 149
update_rmc_tokens, 149
noCrc
LoRaClass, 131
NONE
protocol.h, 263, 264
NOT_ALLOWED
protocol.h, 264
NUM_CALIB_PARAMS
BME280, 70

OCT
Print.h, 260
on_receive
communication.h, 217
receive.cpp, 271
onCadDone
LoRaClass, 123
onDio0Rise
LoRaClass, 140
ONE_TIME_HIGH_RES_MODE
BH1750, 56
ONE_TIME_HIGH_RES_MODE_2
BH1750, 56
ONE_TIME_LOW_RES_MODE
BH1750, 56
onReceive
LoRaClass, 123
onTxDone
LoRaClass, 123
operation_type_to_string
protocol.h, 264
utils_converters.cpp, 280
OperationType
protocol.h, 263
operationType
Frame, 100
outgoing
communication.cpp, 214
OVERCHARGE
event_manager.h, 292

PA_BOOST
LoRa-RP2040.cpp, 241
PA_OUTPUT_PA_BOOST_PIN
pin_config.h, 312
PA_OUTPUT_RFO_PIN
pin_config.h, 312
packetFrequencyError
LoRaClass, 119
packetRssi
LoRaClass, 119
packetSnr
LoRaClass, 119
PARAM_UNECESSARY
protocol.h, 264
parse_packet
LoRaClass, 118
PASS_THROUGH_END
event_manager.h, 292
PASS_THROUGH_START
event_manager.h, 292
peek
LoRaClass, 122
pin_config.cpp
lora_address_local, 306
lora_address_remote, 306
lora_cs_pin, 305
lora_irq_pin, 306
lora_reset_pin, 305
pin_config.h
BUFFER_SIZE, 309
DEBUG_UART_BAUD_RATE, 308
DEBUG_UART_PORT, 308
DEBUG_UART_RX_PIN, 308
DEBUG_UART_TX_PIN, 308
GPS_POWER_ENABLE_PIN, 309
GPS_UART_BAUD_RATE, 309
GPS_UART_PORT, 309
GPS_UART_RX_PIN, 309
GPS_UART_TX_PIN, 309
lora_address_local, 312
lora_address_remote, 313
lora_cs_pin, 312
LORA_DEFAULT_DIO0_PIN, 312
LORA_DEFAULT_RESET_PIN, 311
LORA_DEFAULT_SPI, 311
LORA_DEFAULT_SPI_FREQUENCY, 311
LORA_DEFAULT_SS_PIN, 311
lora_irq_pin, 312
lora_reset_pin, 312
MAIN_I2C_PORT, 308
MAIN_I2C_SCL_PIN, 309
MAIN_I2C_SDA_PIN, 309
PA_OUTPUT_PA_BOOST_PIN, 312
PA_OUTPUT_RFO_PIN, 312
READ_BIT, 311
SD_CARD_DETECT_PIN, 310
SD_CS_PIN, 310
SD_MISO_PIN, 310
SD_MOSI_PIN, 310
SD_SCK_PIN, 310
SD_SPI_PORT, 310
SPI_PORT, 311
SX1278_CS, 310
SX1278_MISO, 310
SX1278_MOSI, 311
SX1278_SCK, 311
POWER
event_manager.h, 291
Power Commands, 27
handle_get_current_charge_solar, 30
handle_get_current_charge_total, 31
handle_get_current_charge_usb, 29
handle_get_current_draw, 32
handle_get_power_manager_ids, 27

handle_get_voltage_5v, 28
handle_get_voltage_battery, 28
POWER_FALLING
event_manager.h, 292
POWER_NORMAL
event_manager.h, 292
POWER_OFF
event_manager.h, 292
POWER_ON
BH1750, 56
event_manager.h, 292
PowerEvent
event_manager.h, 291
powerman_mutex_
PowerManager, 155
PowerManager, 150
charging_solar_active_, 155
charging_usb_active_, 156
check_power_alerts, 154
configure, 153
FALL_RATE_THRESHOLD, 155
FALLING_TREND_REQUIRED, 155
get_current_charge_solar, 151
get_current_charge_total, 152
get_current_charge_usb, 152
get_current_draw, 152
get_voltage_5v, 152
get_voltage_battery, 152
ina3221_, 155
initialize, 151
initialized_, 155
is_charging_solar, 153
is_charging_usb, 153
powerman_mutex_, 155
PowerManager, 151
read_device_ids, 151
SOLAR_CURRENT_THRESHOLD, 154
USB_CURRENT_THRESHOLD, 154
VOLTAGE_LOW_THRESHOLD, 155
VOLTAGE_OVERCHARGE_THRESHOLD, 155
powerManager
main.cpp, 377
PRESSURE
ISensor.h, 352
Print, 156
availableForWrite, 162
clearWriteError, 160
flush, 175
getWriteError, 160
Print, 157
print, 163–168
printFloat, 159
println, 169–174
printNumber, 158
printULLNumber, 158
setWriteError, 159
write, 160–162
write_error, 175
print
Print, 163–168
Print.h
BIN, 260
DEC, 259
HEX, 259
OCT, 260
printFloat
Print, 159
println
Print, 169–174
printNumber
Print, 158
printULLNumber
Print, 158
protocol.h
ANS, 263
BOOL, 263
CommandAccessLevel, 263
DATETIME, 263
DELIMITER, 269
ERR, 263
ERROR, 263
exception_type_to_string, 264
ExceptionType, 263
ExecutionResult, 262
FRAME_BEGIN, 269
FRAME_END, 269
GET, 263
hex_string_to_bytes, 266
INF, 263
INFO, 263
Interface, 264
INVALID_OPERATION, 264
INVALID_PARAM, 264
LORA, 264
MILIAMP, 263
NONE, 263, 264
NOT_ALLOWED, 264
operation_type_to_string, 264
OperationType, 263
PARAM_UNECESSARY, 264
READ_ONLY, 263
READ_WRITE, 263
SECOND, 263
SET, 263
string_to_operation_type, 266
SUCCESS, 263
TEXT, 263
UART, 264
UNDEFINED, 263
value_unit_type_to_string, 267
ValueUnit, 263
VOLT, 263
WRITE_ONLY, 263
pwr_valid_alert
INA3221::masken_reg_t, 143

RADIO_ERROR

event_manager.h, 292
RADIO_INIT
event_manager.h, 292
random
LoRaClass, 131
read
HMC5883L, 102
LoRaClass, 122
READ_BIT
pin_config.h, 311
read_data
BH1750Wrapper, 59
BME280Wrapper, 75
HMC5883LWrapper, 105
ISensor, 113
MPU6050Wrapper, 147
read_device_ids
PowerManager, 151
READ_ONLY
protocol.h, 263
read_raw_all
BME280, 63
read_register
Configuration Functions, 40
HMC5883L, 103
read_temperature
DS3231, 80
READ_WRITE
protocol.h, 263
readRegister
LoRaClass, 136
receive
LoRaClass, 124
receive.cpp
handle_uart_input, 271
on_receive, 271
receive_ack
storage_commands_utils.cpp, 210
storage_commands_utils.h, 212
REG_CONFIG
BME280, 65
REG_CTRL_HUM
BME280, 65
REG_CTRL_MEAS
BME280, 65
REG_DETECTION_OPTIMIZE
LoRa-RP2040.cpp, 239
REG_DETECTION_THRESHOLD
LoRa-RP2040.cpp, 239
REG_DIG_H1
BME280, 69
REG_DIG_H2
BME280, 69
REG_DIG_H3
BME280, 69
REG_DIG_H4
BME280, 69
REG_DIG_H5
BME280, 70
REG_DIG_H6
BME280, 70
REG_DIG_P1_LSB
BME280, 67
REG_DIG_P1_MSB
BME280, 67
REG_DIG_P2_LSB
BME280, 67
REG_DIG_P2_MSB
BME280, 67
REG_DIG_P3_LSB
BME280, 67
REG_DIG_P3_MSB
BME280, 67
REG_DIG_P4_LSB
BME280, 68
REG_DIG_P4_MSB
BME280, 68
REG_DIG_P5_LSB
BME280, 68
REG_DIG_P5_MSB
BME280, 68
REG_DIG_P6_LSB
BME280, 68
REG_DIG_P6_MSB
BME280, 68
REG_DIG_P7_LSB
BME280, 68
REG_DIG_P7_MSB
BME280, 68
REG_DIG_P8_LSB
BME280, 69
REG_DIG_P8_MSB
BME280, 69
REG_DIG_P9_LSB
BME280, 69
REG_DIG_P9_MSB
BME280, 69
REG_DIG_T1_LSB
BME280, 66
REG_DIG_T1_MSB
BME280, 66
REG_DIG_T2_LSB
BME280, 66
REG_DIG_T2_MSB
BME280, 66
REG_DIG_T3_LSB
BME280, 67
REG_DIG_T3_MSB
BME280, 67
REG_DIO_MAPPING_1
LoRa-RP2040.cpp, 240
REG_FIFO
LoRa-RP2040.cpp, 236
REG_FIFO_ADDR_PTR
LoRa-RP2040.cpp, 237
REG_FIFO_RX_BASE_ADDR

LoRa-RP2040.cpp, 237
REG_FIFO_RX_CURRENT_ADDR
LoRa-RP2040.cpp, 237
REG_FIFO_TX_BASE_ADDR
LoRa-RP2040.cpp, 237
REG_FREQ_ERROR_LSB
LoRa-RP2040.cpp, 239
REG_FREQ_ERROR_MID
LoRa-RP2040.cpp, 239
REG_FREQ_ERROR_MSB
LoRa-RP2040.cpp, 239
REG_FRF_LSB
LoRa-RP2040.cpp, 236
REG_FRF_MID
LoRa-RP2040.cpp, 236
REG_FRF_MSB
LoRa-RP2040.cpp, 236
REG_HUMIDITY_MSB
BME280, 66
REG_INVERTIQ
LoRa-RP2040.cpp, 239
REG_INVERTIQ2
LoRa-RP2040.cpp, 240
REG_IRQ_FLAGS
LoRa-RP2040.cpp, 237
REG_LNA
LoRa-RP2040.cpp, 237
REG_MODEM_CONFIG_1
LoRa-RP2040.cpp, 238
REG_MODEM_CONFIG_2
LoRa-RP2040.cpp, 238
REG_MODEM_CONFIG_3
LoRa-RP2040.cpp, 239
REG_OCP
LoRa-RP2040.cpp, 236
REG_OP_MODE
LoRa-RP2040.cpp, 236
REG_PA_CONFIG
LoRa-RP2040.cpp, 236
REG_PA_DAC
LoRa-RP2040.cpp, 240
REG_PAYLOAD_LENGTH
LoRa-RP2040.cpp, 238
REG_PKT_RSSI_VALUE
LoRa-RP2040.cpp, 238
REG_PKT_SNR_VALUE
LoRa-RP2040.cpp, 238
REG_PREAMBLE_LSB
LoRa-RP2040.cpp, 238
REG_PREAMBLE_MSB
LoRa-RP2040.cpp, 238
REG_PRESSURE_MSB
BME280, 66
REG_RESET
BME280, 66
REG_RSSI_VALUE
LoRa-RP2040.cpp, 238
REG_RSSI_WIDEBAND
LoRa-RP2040.cpp, 239
REG_RX_NB_BYTES
LoRa-RP2040.cpp, 237
REG_SYNC_WORD
LoRa-RP2040.cpp, 240
REG_TEMPERATURE_MSB
BME280, 66
REG_VERSION
LoRa-RP2040.cpp, 240
reserved
INA3221::masken_reg_t, 145
RESET
BH1750, 56
reset
BME280, 63
Configuration Functions, 38
INA3221::conf_reg_t, 77
RF_MID_BAND_THRESHOLD
LoRa-RP2040.cpp, 242
rmc_mutex_
NMEAData, 149
rmc_tokens_
NMEAData, 149
rssi
LoRaClass, 120
RSSI_OFFSET_HF_PORT
LoRa-RP2040.cpp, 242
RSSI_OFFSET_LF_PORT
LoRa-RP2040.cpp, 242

SATURDAY
DS3231.h, 189
save_to_storage
EventManager, 94
EventManagerImpl, 98
SD_CARD_DETECT_PIN
pin_config.h, 310
sd_card_mounted
storage.cpp, 358
storage.h, 361
SD_CS_PIN
pin_config.h, 310
SD_MISO_PIN
pin_config.h, 310
SD_MOSI_PIN
pin_config.h, 310
SD_SCK_PIN
pin_config.h, 310
SD_SPI_PORT
pin_config.h, 310
SECOND
protocol.h, 263
seconds
ds3231_data_t, 86
send.cpp
send_frame, 276
send_frame_lora, 275
send_frame_uart, 275
send_message, 274

split_and_send_message, 277
send_data_block
storage_commands_utils.cpp, 210
storage_commands_utils.h, 212
send_frame
communication.h, 219
send.cpp, 276
send_frame_lora
communication.h, 220
send.cpp, 275
send_frame_uart
communication.h, 219
send.cpp, 275
send_message
communication.h, 219
send.cpp, 274
sensor_
BH1750Wrapper, 60
BME280Wrapper, 75
HMC5883LWrapper, 106
MPU6050Wrapper, 147
sensor_configure
SensorWrapper, 178
sensor_init
SensorWrapper, 177
sensor_read_data
SensorWrapper, 178
SensorDataTypeIdentifier
ISensor.h, 352
sensors
SensorWrapper, 179
SensorType
ISensor.h, 351
SensorWrapper, 176
get_instance, 177
sensor_configure, 178
sensor_init, 177
sensor_read_data, 178
sensors, 179
SensorWrapper, 177
SET
protocol.h, 263
set_alert_latch
Alert Functions, 53
set_averaging_mode
Configuration Functions, 44
set_bus_conversion_time
Configuration Functions, 45
set_bus_measurement_disable
Configuration Functions, 44
set_bus_measurement_enable
Configuration Functions, 43
set_crit_alert_limit
Alert Functions, 49
set_mode_continuous
Configuration Functions, 41
set_mode_power_down
Configuration Functions, 41
set_mode_triggered
Configuration Functions, 42
set_pins
LoRaClass, 132
set_power_valid_limit
Alert Functions, 50
set_shunt_conversion_time
Configuration Functions, 45
set_shunt_measurement_disable
Configuration Functions, 43
set_shunt_measurement_enable
Configuration Functions, 42
set_time
DS3231, 79
set_unix_time
DS3231, 80
set_warn_alert_limit
Alert Functions, 49
setCodingRate4
LoRaClass, 127
setFrequency
LoRaClass, 126
setGain
LoRaClass, 130
setLdoFlag
LoRaClass, 136
setOCP
LoRaClass, 130
setPreambleLength
LoRaClass, 128
setSignalBandwidth
LoRaClass, 127
setSPI
LoRaClass, 132
setSPIFrequency
LoRaClass, 132
setSpreadingFactor
LoRaClass, 127
setSyncWord
LoRaClass, 128
setTxPower
LoRaClass, 126
setWriteError
Print, 159
shunt_conv_time
INA3221::conf_reg_t, 77
shunt_sum_alert
INA3221::masken_reg_t, 144
shunt_sum_en_ch1
INA3221::masken_reg_t, 145
shunt_sum_en_ch2
INA3221::masken_reg_t, 145
shunt_sum_en_ch3
INA3221::masken_reg_t, 144
SHUNT_VOLTAGE_LSB_UV
INA3221.h, 322
SHUTDOWN
event_manager.h, 291

SILENT
utils.h, 371
singleTransfer
LoRaClass, 139
sleep
LoRaClass, 125
SOLAR_ACTIVE
event_manager.h, 292
SOLAR_CURRENT_THRESHOLD
PowerManager, 154
SOLAR_INACTIVE
event_manager.h, 292
SPI_PORT
pin_config.h, 311
split_and_send_message
communication.h, 221
send.cpp, 277
splitString
gps_collector.cpp, 298
START_COMMAND
storage_commands.cpp, 206
Storage Commands, 33
handle_file_download, 33
handle_list_files, 34
handle_mount, 35
storage.cpp
fs_init, 357
sd_card_mounted, 358
storage.h
fs_close_file, 361
fs_file_exists, 361
fs_init, 359
fs_open_file, 360
fs_read_file, 360
fs_write_file, 360
sd_card_mounted, 361
storage_commands.cpp
DATA_COMMAND, 206
END_COMMAND, 206
LIST_FILES_COMMAND, 207
MAX_BLOCK_SIZE, 206
MOUNT_COMMAND, 207
START_COMMAND, 206
STORAGE_GROUP, 206
storage_commands_utils.cpp
calculate_checksum, 209
receive_ack, 210
send_data_block, 210
storage_commands_utils.h
calculate_checksum, 212
receive_ack, 212
send_data_block, 212
STORAGE_GROUP
storage_commands.cpp, 206
string_to_operation_type
protocol.h, 266
utils_converters.cpp, 281
SUCCESS
protocol.h, 263
SUNDAY
DS3231.h, 189
SX1278_CS
pin_config.h, 310
SX1278_MISO
pin_config.h, 310
SX1278_MOSI
pin_config.h, 311
SX1278_SCK
pin_config.h, 311
SYSTEM
event_manager.h, 291
systemClock
Clock Management Commands, 14
event_manager.cpp, 287
main.cpp, 377
SystemEvent
event_manager.h, 291

t_fine
BME280, 65
TEMPERATURE
ISensor.h, 352
test/comms/test_converters.cpp, 379, 382
test/comms/test_frame_build.cpp, 383, 385
test/comms/test_frame_coding.cpp, 386, 388
test/comms/test_frame_common.h, 388, 390
test/test_runner.cpp, 390, 397
test_converters.cpp
test_exception_type_conversion, 381
test_hex_string_conversion, 382
test_operation_type_conversion, 380
test_value_unit_type_conversion, 380
test_exception_type_conversion
test_converters.cpp, 381
test_runner.cpp, 395
test_frame_build.cpp
test_frame_build_error, 384
test_frame_build_info, 384
test_frame_build_success, 383
test_frame_build_error
test_frame_build.cpp, 384
test_runner.cpp, 393
test_frame_build_info
test_frame_build.cpp, 384
test_runner.cpp, 393
test_frame_build_success
test_frame_build.cpp, 383
test_runner.cpp, 392
test_frame_coding.cpp
test_frame_decode_basic, 386
test_frame_decode_invalid_header, 387
test_frame_encode_basic, 386
test_frame_common.h
create_test_frame, 389
test_frame_decode_basic
test_frame_coding.cpp, 386
test_runner.cpp, 391

test_frame_decode_invalid_header
test_frame_coding.cpp, 387
test_runner.cpp, 392
test_frame_encode_basic
test_frame_coding.cpp, 386
test_runner.cpp, 391
test_hex_string_conversion
test_converters.cpp, 382
test_runner.cpp, 396
test_operation_type_conversion
test_converters.cpp, 380
test_runner.cpp, 394
test_runner.cpp
main, 397
test_exception_type_conversion, 395
test_frame_build_error, 393
test_frame_build_info, 393
test_frame_build_success, 392
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_value_unit_type_conversion
test_converters.cpp, 380
test_runner.cpp, 395
TEXT
protocol.h, 263
THURSDAY
DS3231.h, 189
TIME
clock_commands.cpp, 191
timestamp
event_manager.h, 294
EventLog, 89
TIMEZONE_OFFSET
clock_commands.cpp, 191
timing_ctrl_alert
INA3221::masken_reg_t, 143
to_string
event_manager.h, 293
EventLog, 89
TUESDAY
DS3231.h, 189

UART
protocol.h, 264
UART_ERROR
event_manager.h, 292
uart_mutex
utils.cpp, 367
uart_print
utils.cpp, 365
utils.h, 371
UNCONFIGURED_POWER_DOWN
BH1750, 56
UNDEFINED
protocol.h, 263
unit
Frame, 100
update_gga_tokens
NMEAData, 149
update_rmc_tokens
NMEAData, 149
USB_CONNECTED
event_manager.h, 292
USB_CURRENT_THRESHOLD
PowerManager, 154
USB_DISCONNECTED
event_manager.h, 292
utils.cpp
crc16, 366
g_uart_verbosity, 367
get_level_color, 363
get_level_prefix, 364
uart_mutex, 367
uart_print, 365
utils.h
ANSI_BLUE, 370
ANSI_CYAN, 370
ANSI_GREEN, 370
ANSI_RED, 370
ANSI_RESET, 370
ANSI_YELLOW, 370
crc16, 372
DEBUG, 371
ERROR, 371
EVENT, 371
g_uart_verbosity, 373
INFO, 371
SILENT, 371
uart_print, 371
VerbosityLevel, 370
WARNING, 371
utils_converters.cpp
exception_type_to_string, 279
hex_string_to_bytes, 282
operation_type_to_string, 280
string_to_operation_type, 281
value_unit_type_to_string, 279

value
Frame, 100
value_unit_type_to_string
protocol.h, 267
utils_converters.cpp, 279
ValueUnit
protocol.h, 263
VerbosityLevel
utils.h, 370
VOLT
protocol.h, 263
VOLTAGE_LOW_THRESHOLD
event_manager.cpp, 286
PowerManager, 155
VOLTAGE_OVERCHARGE_THRESHOLD
event_manager.cpp, 286

PowerManager, 155

warn_alert_ch1
INA3221::masken_reg_t, 144
warn_alert_ch2
INA3221::masken_reg_t, 143
warn_alert_ch3
INA3221::masken_reg_t, 143
warn_alert_latch_en
INA3221::masken_reg_t, 144
WARNING
utils.h, 371
WATCHDOG_RESET
event_manager.h, 291
WEDNESDAY
DS3231.h, 189
write
LoRaClass, 120, 121
Print, 160–162
write8
BH1750, 57
write_error
Print, 175
WRITE_ONLY
protocol.h, 263
write_register
HMC5883L, 102
writeIndex
EventManager, 95
writeRegister
LoRaClass, 137

year
ds3231_data_t, 86