

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_ADDRESS	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 SensorDataTypelIdentifier	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 ¶m, OperationType operationType)`

Command ID: 3.3

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

Command ID: 7.2

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

Command ID: 2

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

Command ID: 6.1

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

Command ID: 1

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

Command ID: 0

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

Command ID: 2.5

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

Command ID: 2.6

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

Command ID: 2.4

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

Command ID: 2.7

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

Command ID: 5.2

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

Command ID: 7.4

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

Command ID: 5.1

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

Command ID: 3.7

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

Command ID: 2.0

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

Command ID: 7.3

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

Command ID: 2.3

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

Command ID: 2.2

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

Command ID: 7.1

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

Command ID: 6.0

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

Command ID: 6.4

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

Command ID: 3.0

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

Command ID: 3.1

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

Command ID: 1.8

Chapter 2

Topic Index

2.1 Topics

Here is a list of all topics with brief descriptions:

Clock Management Commands	11
Command System	14
Diagnostic Commands	17
Event Commands	21
GPS Commands	23
Power Commands	27
Storage Commands	33
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 <code>EventManager</code> 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 Kabisat firmware	362
lib/utils.h	Utility functions and definitions for the Kabisat 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 ¶m, OperationType operationType)`
Handler for getting and setting system time.
- `Frame handle_timezone_offset (const std::string ¶m, OperationType operationType)`
Handler for getting and setting timezone offset.
- `Frame handle_clock_sync_interval (const std::string ¶m, OperationType operationType)`
Handler for getting and setting clock synchronization interval.
- `Frame handle_get_last_sync_time (const std::string ¶m, OperationType operationType)`
Handler for getting last clock sync time.

Variables

- `DS3231 systemClock`

6.1.1 Detailed Description

Commands for managing system time and clock settings.

6.1.2 Function Documentation

6.1.2.1 `handle_time()`

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

Handler for getting and setting system time.

Parameters

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

Returns

`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"

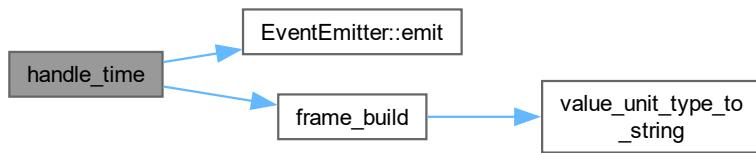
SET: **KBST;0;SET;3;0;TIMESTAMP;KBST**

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

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

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

Returns

Frame containing success/error and sync interval in seconds

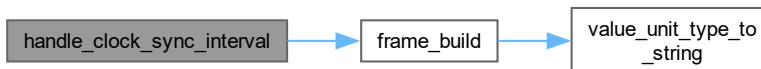
Note

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

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

Variables

- `CommandMap commandHandlers`
Global map of all command handlers.

6.2.1 Detailed Description

Core command system implementation.

6.2.2 Typedef Documentation

6.2.2.1 CommandHandler

```
using CommandHandler = std::function<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

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

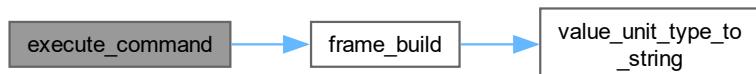
Returns

Frame Response frame containing execution result

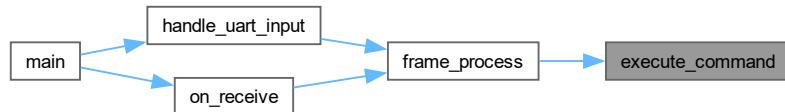
Looks up the command handler in commandHandlers map and executes it

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.2.4 Variable Documentation

6.2.4.1 commandHandlers

CommandMap commandHandlers

Initial value:

```

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

```

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

```

Global map of all command handlers.

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

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

6.3 Diagnostic Commands

Functions

- [Frame handle_get_commands_list](#) (const std::string ¶m, [OperationType](#) operationType)
Handler for listing all available commands on UART.
- [Frame handle_get_build_version](#) (const std::string ¶m, [OperationType](#) operationType)
Get firmware build version.
- [Frame handle_verbosity](#) (const std::string ¶m, [OperationType](#) operationType)
Handles setting or getting the UART verbosity level.
- [Frame handle_enter_bootloader_mode](#) (const std::string ¶m, [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

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

Returns

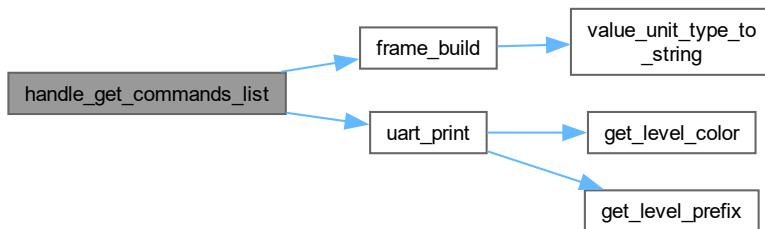
[Frame](#) containing success/error and command list

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

Print all available commands on UART port

Command Command ID: 0Definition 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

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

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

Returns

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.

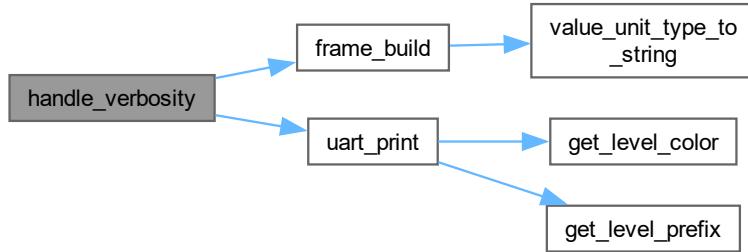
KBST;0;SET;1;8;[level];TSBK - Sets the verbosity level.

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

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

Returns

`Frame` with operation result

Note

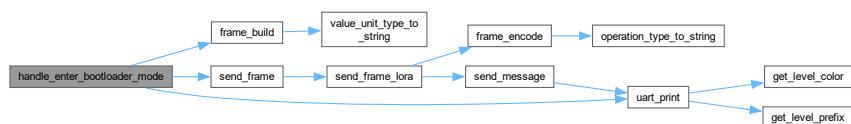
KBST;0;SET;1;9;;TSBK

Reboot the system to USB firmware loader

Command Command ID: 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 ¶m, OperationType operationType)`
Handler for retrieving last N events from the event log.
- `Frame handle_get_event_count (const std::string ¶m, OperationType operationType)`
Handler for getting total number of events in the log.

6.4.1 Detailed Description

Commands for accessing and managing system event logs.

6.4.2 Function Documentation

6.4.2.1 handle_get_last_events()

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

Handler for retrieving last N events from the event log.

Parameters

<code>param</code>	Number of events to retrieve (optional, default 10)
<code>operationType</code>	GET

Returns

`Frame` containing:

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

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

Returns up to 10 most recent events

Command Command ID: 5.1Definition 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

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

Returns**Frame** containing:

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

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

Returns the total number of events in the log

Command Command ID: 5.2Definition 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 ¶m, OperationType operationType)`
Handler for controlling GPS module power state.
- `Frame handle_enable_gps_uart_passthrough (const std::string ¶m, OperationType operationType)`
Handler for enabling GPS transparent mode (UART pass-through)
- `Frame handle_get_rmc_data (const std::string ¶m, OperationType operationType)`
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- `Frame handle_get_gga_data (const std::string ¶m, OperationType operationType)`
Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.

6.5.1 Detailed Description

Commands for controlling and monitoring the GPS module.

6.5.2 Function Documentation

6.5.2.1 handle_gps_power_status()

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

Handler for controlling GPS module power state.

Parameters

<code>param</code>	For SET: "0" to power off, "1" to power on. For GET: empty
<code>operationType</code>	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**

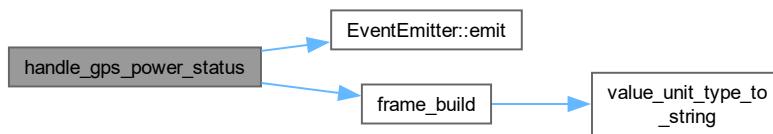
Return current GPS module power state: ON/OFF

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

POWER - 0 - OFF, 1 - ON

Command Command ID: 7.1Definition 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

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

Returns**Frame** containing:

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

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

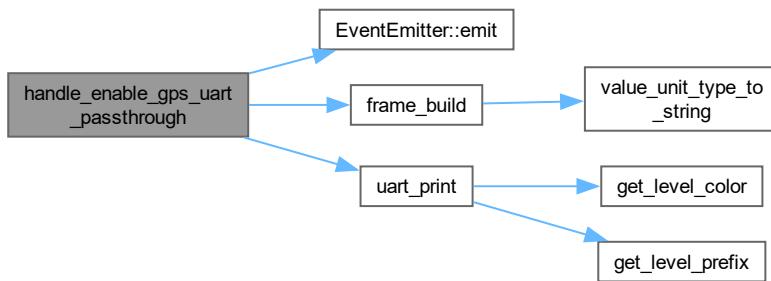
TIMEOUT - 1-600s, default 60s

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

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

Command Command ID: 7.2Definition 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

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

<i>param</i>	Empty string expected
<i>operationType</i>	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 ¶m, OperationType operationType)`
Handler for retrieving Power Manager IDs.
- `Frame handle_get_voltage_battery (const std::string ¶m, OperationType operationType)`
Handler for getting battery voltage.
- `Frame handle_get_voltage_5v (const std::string ¶m, OperationType operationType)`
Handler for getting 5V rail voltage.
- `Frame handle_get_current_charge_usb (const std::string ¶m, OperationType operationType)`
Handler for getting USB charge current.
- `Frame handle_get_current_charge_solar (const std::string ¶m, OperationType operationType)`
Handler for getting solar panel charge current.
- `Frame handle_get_current_charge_total (const std::string ¶m, OperationType operationType)`
Handler for getting total charge current.
- `Frame handle_get_current_draw (const std::string ¶m, OperationType operationType)`
Handler for getting system current draw.

6.6.1 Detailed Description

Commands for monitoring power subsystem and battery management.

6.6.2 Function Documentation

6.6.2.1 handle_get_power_manager_ids()

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

Handler for retrieving Power Manager IDs.

Parameters

<code>param</code>	Empty string expected
<code>operationType</code>	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: 2.0Definition 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

<i>param</i>	Empty string expected
<i>operationType</i>	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: 2.2Definition 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

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

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

Returns

Frame containing:

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

Note

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

This command is used to retrieve the USB charge current

Command Command ID: 2.4

Definition at line 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

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

Returns

Frame containing:

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

Note

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

This command is used to retrieve the solar panel charge current

Command Command ID: 2.5

Definition at line 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

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

Returns

Frame containing:

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

Note

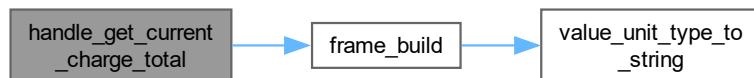
KBST;0;GET;2;6;;TSBK

This command is used to retrieve the total charge current

Command Command ID: 2.6

Definition at line 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

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

Returns

Frame containing:

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

Note

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

This command is used to retrieve the system current draw

Command Command ID: 2.7

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

6.7.1 Detailed Description

Commands for interacting with the SD card storage.

6.7.2 Function Documentation

6.7.2.1 handle_file_download()

```
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

<code>param</code>	The filename to download.
<code>operationType</code>	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

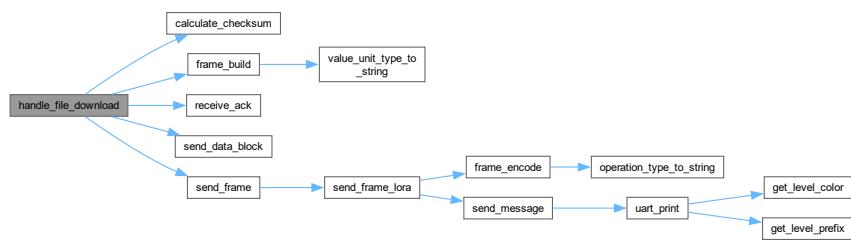
KBST;0;GET;6;1;[filename];TSBK

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

<code>param</code>	Unused.
<code>operationType</code>	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

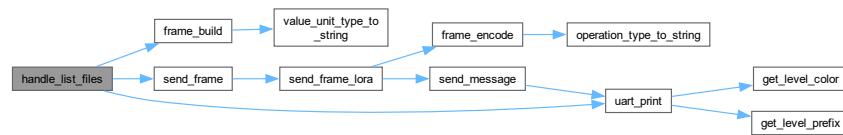
KBST;0;GET;6;0;;TSBK

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

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

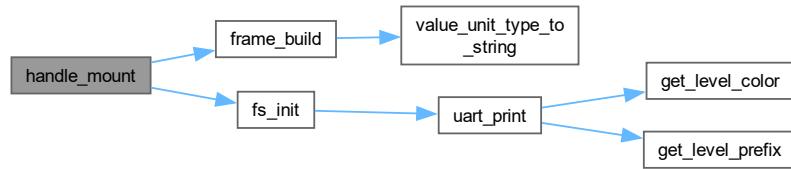
KBST;0;SET;6;4;[0|1];TSBK

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

Command Command ID: 6.4

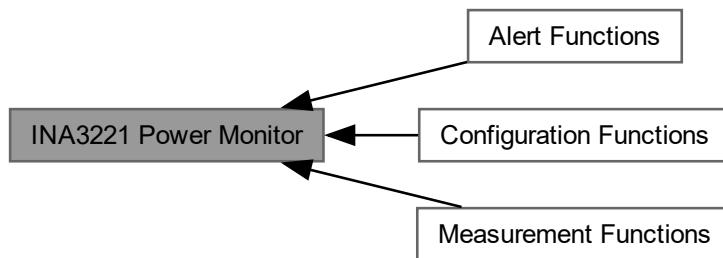
Definition at line 180 of file [storage_commands.cpp](#).

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

<i>addr</i>	I2C address of the device
<i>i2c</i>	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.

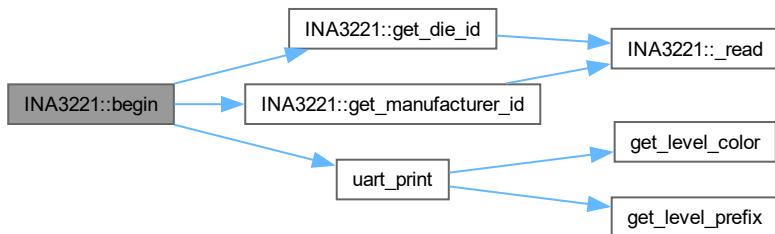
Returns

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()

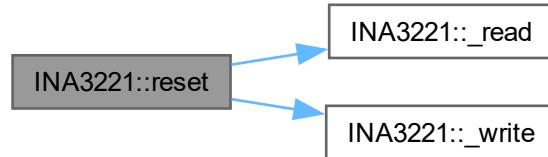
```
void INA3221::reset ()
```

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 ()
```

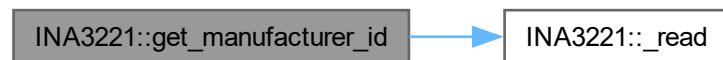
Get the manufacturer ID of the device.

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

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

Returns

16-bit value read from the register

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

Here is the call graph for this function:

**6.8.2.2.7 set_mode_power_down()**

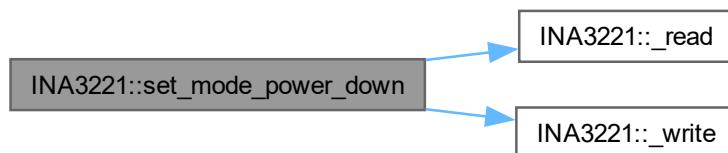
```
void INA3221::set_mode_power_down ()
```

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()**

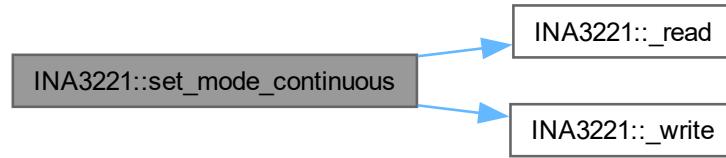
```
void INA3221::set_mode_continuous ()
```

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()`

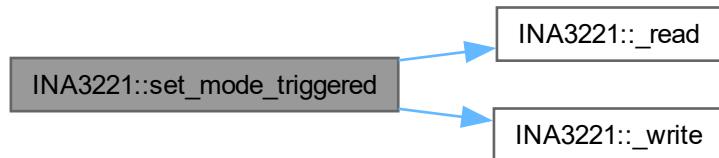
```
void INA3221::set_mode_triggered ()
```

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:



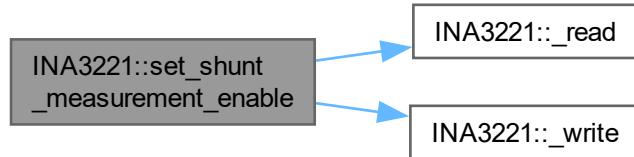
6.8.2.2.10 `set_shunt_measurement_enable()`

```
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:



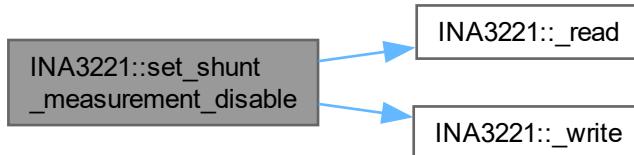
6.8.2.2.11 `set_shunt_measurement_disable()`

```
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:



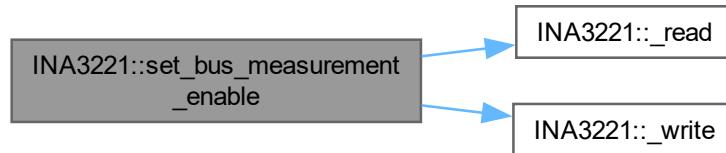
6.8.2.2.12 `set_bus_measurement_enable()`

```
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:



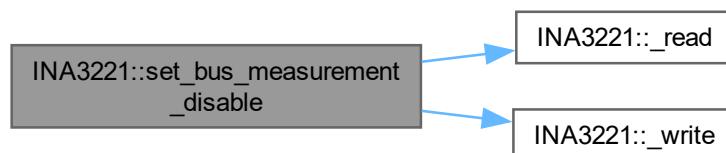
6.8.2.2.13 `set_bus_measurement_disable()`

```
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)
```

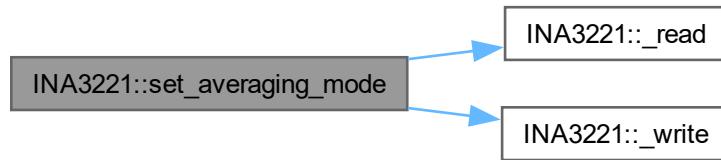
Set the averaging mode for measurements.

Parameters

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

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

Here is the call graph for this function:



6.8.2.2.15 set_bus_conversion_time()

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

Set bus voltage conversion time.

Parameters

<code>convTime</code>	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)
```

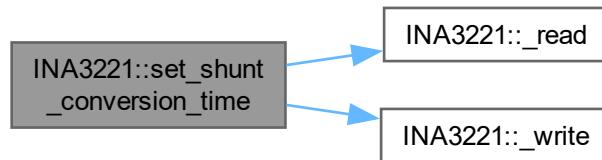
Set shunt voltage conversion time.

Parameters

<code>convTime</code>	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

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

Returns

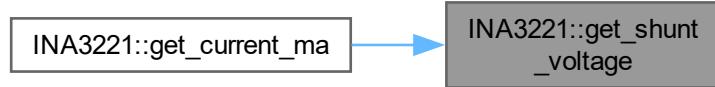
Shunt voltage in microvolts (μ V)

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

Here is the call graph for this function:



Here is the caller graph for this function:



6.8.3.2.2 `get_current_ma()`

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

Get current for a specific channel.

Parameters

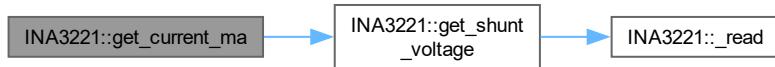
<i>channel</i>	Channel number (1-3)
----------------	----------------------

Returns

Current in millamps (mA)

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

Here is the call graph for this function:

**6.8.3.2.3 get_voltage()**

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

Get bus voltage for a specific channel.

Parameters

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

Returns

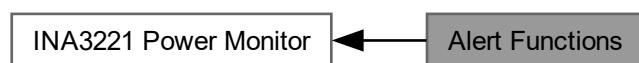
Voltage in volts (V)

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

Here is the call graph for this function:

**6.8.4 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

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

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

Here is the call graph for this function:



6.8.4.2.2 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

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

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

Here is the call graph for this function:



6.8.4.2.3 set_power_valid_limit()

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

Set power valid voltage range.

Parameters

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

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

Here is the call graph for this function:



6.8.4.2.4 enable_alerts()

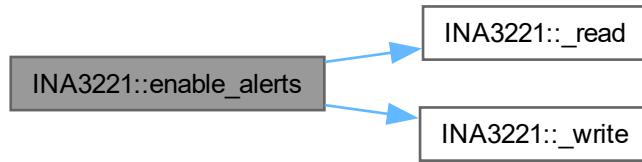
```
void INA3221::enable_alerts ()
```

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

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

Returns

true if warning alert is active, false otherwise

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

Here is the call graph for this function:



6.8.4.2.6 `get_crit_alert()`

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

Get critical alert status for a channel.

Parameters

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

Returns

true if critical alert is active, false otherwise

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

Here is the call graph for this function:

**6.8.4.2.7 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)
```

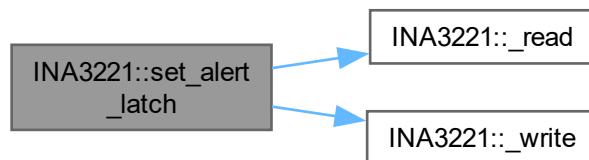
Set alert latch mode.

Parameters

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

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

Here is the call graph for this function:



Chapter 7

Class Documentation

7.1 BH1750 Class Reference

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

Public Types

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

Public Member Functions

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

Private Member Functions

- void `write8` (`uint8_t data`)

Private Attributes

- `uint8_t _i2c_addr`

7.1.1 Detailed Description

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

7.1.2 Member Enumeration Documentation

7.1.2.1 Mode

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

Enumerator

UNCONFIGURED_POWER_DOWN	
POWER_ON	
RESET	
CONTINUOUS_HIGH_RES_MODE	
CONTINUOUS_HIGH_RES_MODE_2	
CONTINUOUS_LOW_RES_MODE	
ONE_TIME_HIGH_RES_MODE	
ONE_TIME_HIGH_RES_MODE_2	
ONE_TIME_LOW_RES_MODE	

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

7.1.3 Constructor & Destructor Documentation

7.1.3.1 BH1750()

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

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

7.1.4 Member Function Documentation

7.1.4.1 begin()

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

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

Here is the call graph for this function:



7.1.4.2 configure()

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.1.4.3 get_light_level()

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

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

7.1.4.4 write8()

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

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

Here is the caller graph for this function:



7.1.5 Member Data Documentation

7.1.5.1 _i2c_addr

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

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

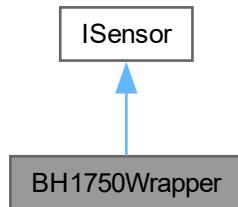
The documentation for this class was generated from the following files:

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

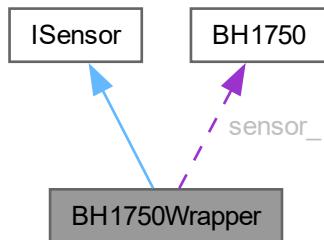
7.2 BH1750Wrapper Class Reference

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

Inheritance diagram for BH1750Wrapper:



Collaboration diagram for BH1750Wrapper:



Public Member Functions

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

Public Member Functions inherited from [ISensor](#)

- `virtual ~ISensor ()=default`

Private Attributes

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

7.2.1 Detailed Description

Definition at line 9 of file [BH1750_WRAPPER.h](#).

7.2.2 Constructor & Destructor Documentation

7.2.2.1 `BH1750Wrapper()`

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

Definition at line 6 of file [BH1750_WRAPPER.cpp](#).

7.2.3 Member Function Documentation

7.2.3.1 `get_i2c_addr()`

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

7.2.3.2 `init()`

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

Implements [ISensor](#).

Definition at line 10 of file [BH1750_WRAPPER.cpp](#).

7.2.3.3 `read_data()`

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

Implements [ISensor](#).

Definition at line 15 of file [BH1750_WRAPPER.cpp](#).

7.2.3.4 `is_initialized()`

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

Implements [ISensor](#).

Definition at line 22 of file [BH1750_WRAPPER.cpp](#).

7.2.3.5 `get_type()`

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

Implements [ISensor](#).

Definition at line 26 of file [BH1750_WRAPPER.cpp](#).

7.2.3.6 `configure()`

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

Implements [ISensor](#).

Definition at line 30 of file [BH1750_WRAPPER.cpp](#).

7.2.4 Member Data Documentation

7.2.4.1 `sensor_`

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

Definition at line 11 of file [BH1750_WRAPPER.h](#).

7.2.4.2 `initialized_`

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

Definition at line 12 of file [BH1750_WRAPPER.h](#).

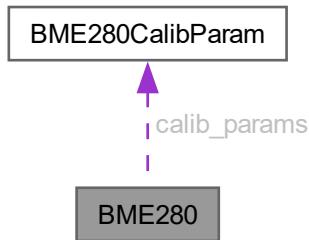
The documentation for this class was generated from the following files:

- lib/sensors/BH1750/[BH1750_WRAPPER.h](#)
- lib/sensors/BH1750/[BH1750_WRAPPER.cpp](#)

7.3 BME280 Class Reference

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

Collaboration diagram for BME280:



Public Member Functions

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

Static Public Attributes

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

Private Member Functions

- `bool configure_sensor ()`
- `bool get_calibration_parameters ()`

Private Attributes

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

Static Private Attributes

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

7.3.1 Detailed Description

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

7.3.2 Constructor & Destructor Documentation

7.3.2.1 BME280()

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

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

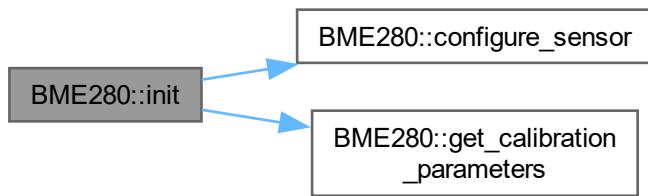
7.3.3 Member Function Documentation

7.3.3.1 init()

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

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

Here is the call graph for this function:



7.3.3.2 reset()

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

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

7.3.3.3 read_raw_all()

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

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

7.3.3.4 convert_temperature()

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

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

7.3.3.5 convert_pressure()

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

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

7.3.3.6 convert_humidity()

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

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

7.3.3.7 configure_sensor()

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

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

Here is the caller graph for this function:



7.3.3.8 get_calibration_parameters()

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

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

Here is the caller graph for this function:



7.3.4 Member Data Documentation

7.3.4.1 ADDR_SDO_LOW

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

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

7.3.4.2 ADDR_SDO_HIGH

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

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

7.3.4.3 i2c_port

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

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

7.3.4.4 device_addr

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

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

7.3.4.5 calib_params

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

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

7.3.4.6 initialized_

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

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

7.3.4.7 t_fine

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

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

7.3.4.8 REG_CONFIG

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

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

7.3.4.9 REG_CTRL_MEAS

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

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

7.3.4.10 REG_CTRL_HUM

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

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

7.3.4.11 REG_RESET

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

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

7.3.4.12 REG_PRESSURE_MSB

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

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

7.3.4.13 REG_TEMPERATURE_MSB

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

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

7.3.4.14 REG_HUMIDITY_MSB

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

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

7.3.4.15 REG_DIG_T1_LSB

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

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

7.3.4.16 REG_DIG_T1_MSB

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

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

7.3.4.17 REG_DIG_T2_LSB

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

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

7.3.4.18 REG_DIG_T2_MSB

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

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

7.3.4.19 REG_DIG_T3_LSB

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

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

7.3.4.20 REG_DIG_T3_MSB

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

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

7.3.4.21 REG_DIG_P1_LSB

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

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

7.3.4.22 REG_DIG_P1_MSB

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

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

7.3.4.23 REG_DIG_P2_LSB

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

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

7.3.4.24 REG_DIG_P2_MSB

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

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

7.3.4.25 REG_DIG_P3_LSB

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

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

7.3.4.26 REG_DIG_P3_MSB

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

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

7.3.4.27 REG_DIG_P4_LSB

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

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

7.3.4.28 REG_DIG_P4_MSB

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

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

7.3.4.29 REG_DIG_P5_LSB

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

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

7.3.4.30 REG_DIG_P5_MSB

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

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

7.3.4.31 REG_DIG_P6_LSB

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

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

7.3.4.32 REG_DIG_P6_MSB

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

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

7.3.4.33 REG_DIG_P7_LSB

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

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

7.3.4.34 REG_DIG_P7_MSB

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

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

7.3.4.35 REG_DIG_P8_LSB

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

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

7.3.4.36 REG_DIG_P8_MSB

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

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

7.3.4.37 REG_DIG_P9_LSB

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

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

7.3.4.38 REG_DIG_P9_MSB

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

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

7.3.4.39 REG_DIG_H1

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

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

7.3.4.40 REG_DIG_H2

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

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

7.3.4.41 REG_DIG_H3

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

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

7.3.4.42 REG_DIG_H4

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

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

7.3.4.43 REG_DIG_H5

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

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

7.3.4.44 REG_DIG_H6

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

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

7.3.4.45 NUM_CALIB_PARAMS

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

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

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

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

7.4 BME280CalibParam Struct Reference

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

Public Attributes

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

7.4.1 Detailed Description

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

7.4.2 Member Data Documentation

7.4.2.1 dig_t1

```
uint16_t BME280CalibParam::dig_t1
```

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

7.4.2.2 dig_t2

```
int16_t BME280CalibParam::dig_t2
```

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

7.4.2.3 dig_t3

```
int16_t BME280CalibParam::dig_t3
```

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

7.4.2.4 dig_p1

```
uint16_t BME280CalibParam::dig_p1
```

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

7.4.2.5 dig_p2

```
int16_t BME280CalibParam::dig_p2
```

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

7.4.2.6 dig_p3

```
int16_t BME280CalibParam::dig_p3
```

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

7.4.2.7 `dig_p4`

```
int16_t BME280CalibParam::dig_p4
```

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

7.4.2.8 `dig_p5`

```
int16_t BME280CalibParam::dig_p5
```

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

7.4.2.9 `dig_p6`

```
int16_t BME280CalibParam::dig_p6
```

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

7.4.2.10 `dig_p7`

```
int16_t BME280CalibParam::dig_p7
```

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

7.4.2.11 `dig_p8`

```
int16_t BME280CalibParam::dig_p8
```

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

7.4.2.12 `dig_p9`

```
int16_t BME280CalibParam::dig_p9
```

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

7.4.2.13 `dig_h1`

```
uint8_t BME280CalibParam::dig_h1
```

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

7.4.2.14 `dig_h2`

```
int16_t BME280CalibParam::dig_h2
```

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

7.4.2.15 dig_h3

```
uint8_t BME280CalibParam::dig_h3
```

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

7.4.2.16 dig_h4

```
int16_t BME280CalibParam::dig_h4
```

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

7.4.2.17 dig_h5

```
int16_t BME280CalibParam::dig_h5
```

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

7.4.2.18 dig_h6

```
int8_t BME280CalibParam::dig_h6
```

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

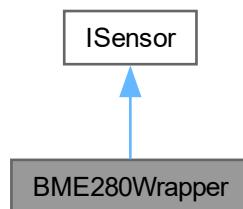
The documentation for this struct was generated from the following file:

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

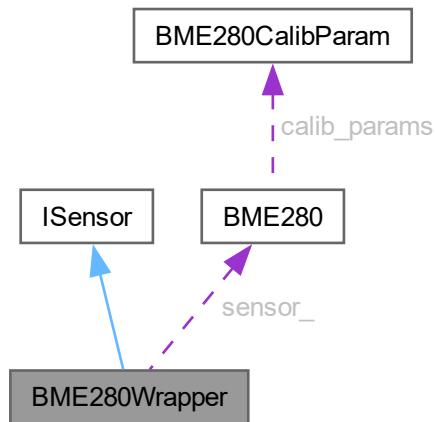
7.5 BME280Wrapper Class Reference

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

Inheritance diagram for BME280Wrapper:



Collaboration diagram for BME280Wrapper:



Public Member Functions

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

Public Member Functions inherited from [ISensor](#)

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

Private Attributes

- [BME280 sensor_](#)
- bool [initialized_ = false](#)

7.5.1 Detailed Description

Definition at line 8 of file [BME280_WRAPPER.h](#).

7.5.2 Constructor & Destructor Documentation

7.5.2.1 [BME280Wrapper\(\)](#)

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

Definition at line 3 of file [BME280_WRAPPER.cpp](#).

7.5.3 Member Function Documentation

7.5.3.1 init()

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

Implements [ISensor](#).

Definition at line 5 of file [BME280_WRAPPER.cpp](#).

7.5.3.2 read_data()

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

Implements [ISensor](#).

Definition at line 10 of file [BME280_WRAPPER.cpp](#).

7.5.3.3 is_initialized()

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

Implements [ISensor](#).

Definition at line 26 of file [BME280_WRAPPER.cpp](#).

7.5.3.4 get_type()

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

Implements [ISensor](#).

Definition at line 30 of file [BME280_WRAPPER.cpp](#).

7.5.3.5 configure()

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

Implements [ISensor](#).

Definition at line 34 of file [BME280_WRAPPER.cpp](#).

7.5.4 Member Data Documentation

7.5.4.1 sensor_

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

Definition at line 10 of file [BME280_WRAPPER.h](#).

7.5.4.2 initialized_

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

Definition at line 11 of file [BME280_WRAPPER.h](#).

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

- lib/sensors/BME280/[BME280_WRAPPER.h](#)
- lib/sensors/BME280/[BME280_WRAPPER.cpp](#)

7.6 INA3221::conf_reg_t Struct Reference

Configuration register bit fields.

Public Attributes

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

7.6.1 Detailed Description

Configuration register bit fields.

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

7.6.2 Member Data Documentation

7.6.2.1 mode_shunt_en

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

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

7.6.2.2 mode_bus_en

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

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

7.6.2.3 mode_continious_en

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

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

7.6.2.4 shunt_conv_time

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

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

7.6.2.5 bus_conv_time

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

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

7.6.2.6 avg_mode

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

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

7.6.2.7 ch3_en

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

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

7.6.2.8 ch2_en

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

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

7.6.2.9 ch1_en

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

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

7.6.2.10 reset

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

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

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

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

7.7 DS3231 Class Reference

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

Public Member Functions

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

Private Member Functions

- int [i2c_read_reg](#) (uint8_t reg_addr, size_t length, uint8_t *data)
Library function to read a specific I2C register address.
- int [i2c_write_reg](#) (uint8_t reg_addr, size_t length, uint8_t *data)
Library function to write to a specific I2C register address.
- uint8_t [bin_to_bcd](#) (const uint8_t data)
Library function that takes an 8-bit unsigned integer and converts it into a Binary Coded Decimal number that can be written to [DS3231](#) registers.
- uint8_t [bcd_to_bin](#) (const uint8_t bcd)
Library function that takes a BCD number and converts it to an unsigned 8-bit integer.

Private Attributes

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

7.7.1 Detailed Description

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

7.7.2 Constructor & Destructor Documentation

7.7.2.1 DS3231()

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

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

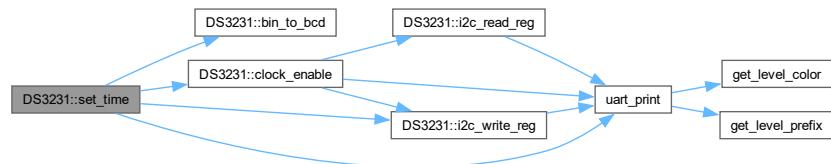
7.7.3 Member Function Documentation

7.7.3.1 set_time()

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

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

Here is the call graph for this function:



Here is the caller graph for this function:

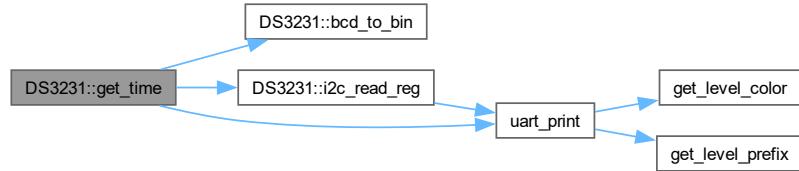


7.7.3.2 get_time()

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

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

Here is the call graph for this function:



Here is the caller graph for this function:

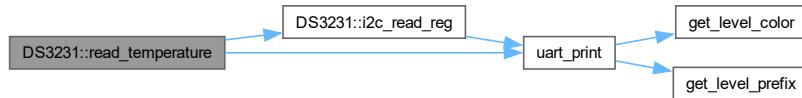


7.7.3.3 read_temperature()

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

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

Here is the call graph for this function:

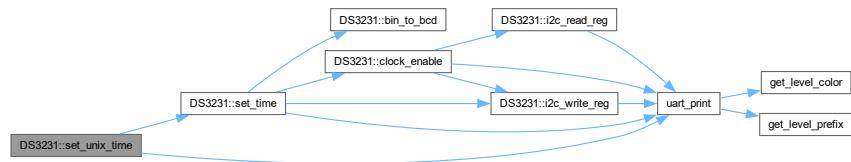


7.7.3.4 set_unix_time()

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

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

Here is the call graph for this function:

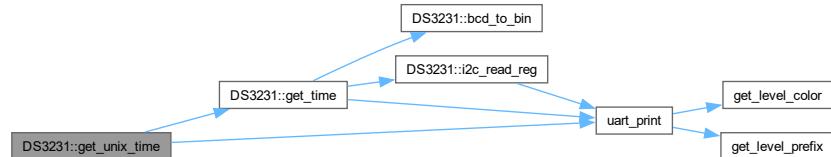


7.7.3.5 get_unix_time()

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

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

Here is the call graph for this function:

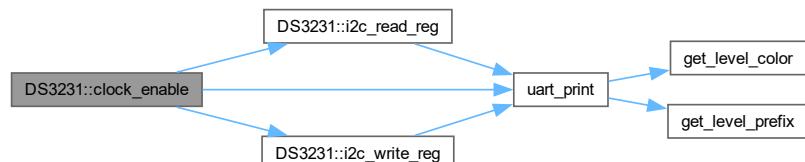


7.7.3.6 clock_enable()

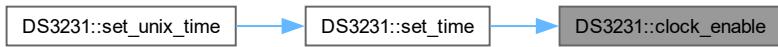
```
int DS3231::clock_enable ()
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.7 i2c_read_reg()

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

Library function to read a specific I2C register address.

Parameters

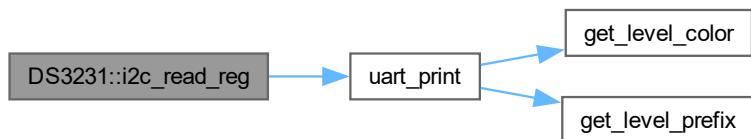
in	<i>reg_addr</i>	Register address to be read.
in	<i>length</i>	length of the data in bytes to be read.
out	<i>data</i>	Buffer to store the read data.

Returns

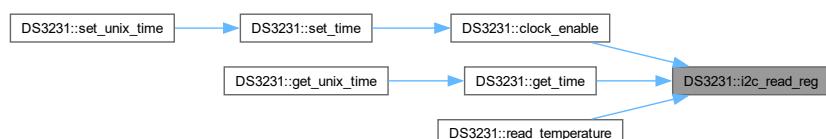
0 if successful, -1 if i2c failure.

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.8 i2c_write_reg()

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

Library function to write to a specific I2C register address.

Parameters

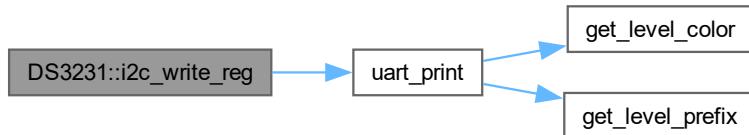
in	<i>reg_addr</i>	Register address to be written.
in	<i>length</i>	Length of the data to be written in bytes.
in	<i>data</i>	Pointer to the data buffer.

Returns

0 if successful, -1 if i2c failure.

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.7.3.9 bin_to_bcd()

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

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

Parameters

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

Returns

Number in BCD form.

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

Here is the caller graph for this function:



7.7.3.10 bcd_to_bin()

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

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

Parameters

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

Returns

Unsigned 8-bit integer.

Definition at line 211 of file [DS3231.cpp](#).

Here is the caller graph for this function:



7.7.4 Member Data Documentation

7.7.4.1 i2c

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

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

7.7.4.2 ds3231_addr

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

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

7.7.4.3 clock_mutex_

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

Definition at line 70 of file [DS3231.h](#).

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

- lib/clock/[DS3231.h](#)
- lib/clock/[DS3231.cpp](#)

7.8 ds3231_data_t Struct Reference

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

Public Attributes

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

7.8.1 Detailed Description

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

7.8.2 Member Data Documentation

7.8.2.1 seconds

```
uint8_t ds3231_data_t::seconds
```

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

7.8.2.2 minutes

```
uint8_t ds3231_data_t::minutes
```

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

7.8.2.3 hours

```
uint8_t ds3231_data_t::hours
```

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

7.8.2.4 day

```
uint8_t ds3231_data_t::day
```

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

7.8.2.5 date

```
uint8_t ds3231_data_t::date
```

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

7.8.2.6 month

```
uint8_t ds3231_data_t::month
```

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

7.8.2.7 year

```
uint8_t ds3231_data_t::year
```

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

7.8.2.8 century

```
bool ds3231_data_t::century
```

Definition at line 45 of file [DS3231.h](#).

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

- lib/clock/[DS3231.h](#)

7.9 EventEmitter Class Reference

Provides a static method for emitting events.

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

Static Public Member Functions

- template<typename T>
static void [emit](#) ([EventGroup group](#), T event)
Emits an event.

7.9.1 Detailed Description

Provides a static method for emitting events.

Definition at line 230 of file [event_manager.h](#).

7.9.2 Member Function Documentation

7.9.2.1 emit()

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

Emits an event.

Parameters

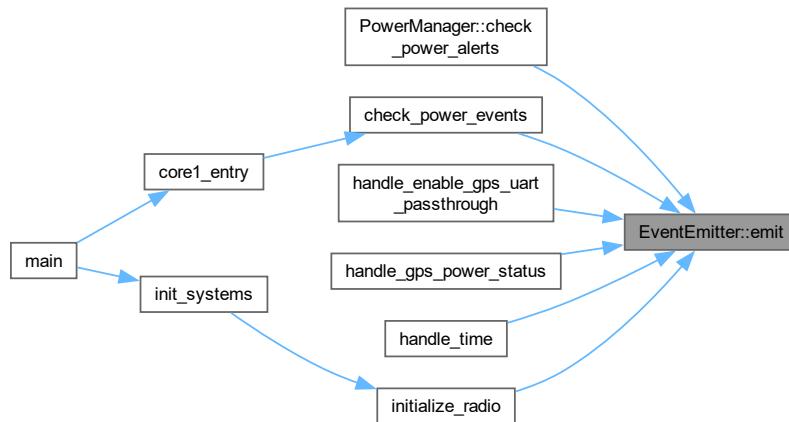
<i>group</i>	The event group.
<i>event</i>	The event identifier.

Template Parameters

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

Definition at line 239 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:

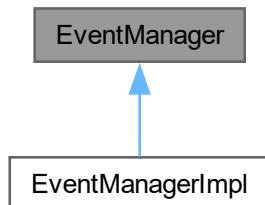
- lib/eventman/[event_manager.h](#)

7.11 EventManager Class Reference

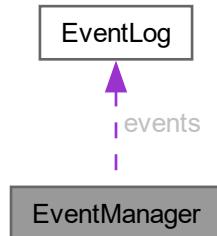
Manages the event logging system.

```
#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()

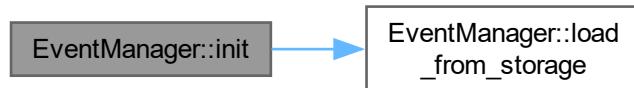
```
virtual void EventManager::init () [inline], [virtual]
```

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.

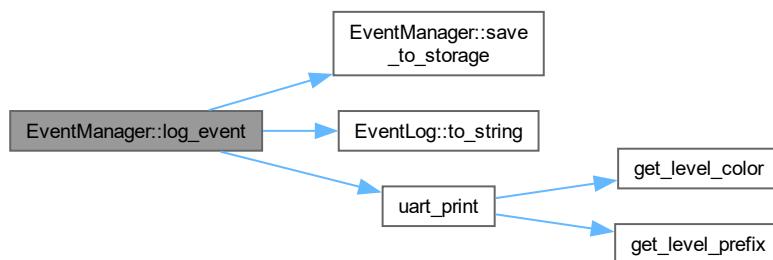
Parameters

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

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

Definition at line 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

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

Returns

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

Parameters

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

Returns

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

Definition at line 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:

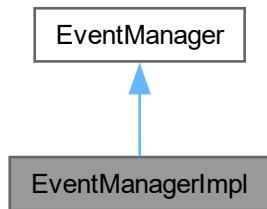
- lib/eventman/[event_manager.h](#)
- lib/eventman/[event_manager.cpp](#)

7.12 EventManagerImpl Class Reference

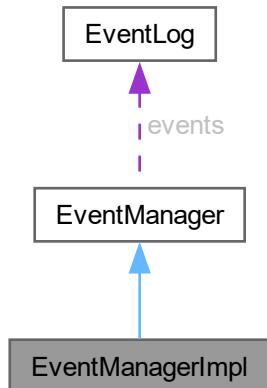
Implementation of the [EventManager](#) class.

```
#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

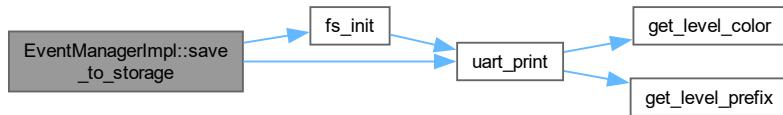
True if the events were successfully saved, false otherwise.

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 214 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](#).

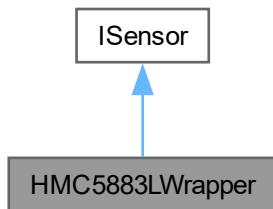
The documentation for this class was generated from the following files:

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

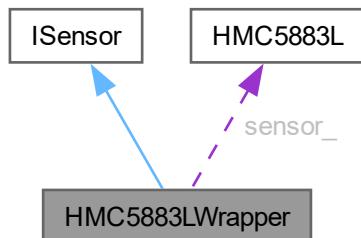
7.16 HMC5883LWrapper Class Reference

```
#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 \(SensorDataTypelIdentifier type\)](#) override
- bool [is_initialized \(\)](#) const override
- [SensorType get_type \(\)](#) const override
- bool [configure \(const std::map< std::string, std::string > &config\)](#) override

Public Member Functions inherited from [ISensor](#)

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

Private Attributes

- `HMC5883L sensor_`
- `bool initialized_`

7.16.1 Detailed Description

Definition at line 7 of file [HMC5883L_WRAPPER.h](#).

7.16.2 Constructor & Destructor Documentation

7.16.2.1 HMC5883LWrapper()

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

Definition at line 5 of file [HMC5883L_WRAPPER.cpp](#).

7.16.3 Member Function Documentation

7.16.3.1 init()

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

Implements [ISensor](#).

Definition at line 7 of file [HMC5883L_WRAPPER.cpp](#).

7.16.3.2 read_data()

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

Implements [ISensor](#).

Definition at line 12 of file [HMC5883L_WRAPPER.cpp](#).

7.16.3.3 is_initialized()

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

Implements [ISensor](#).

Definition at line 35 of file [HMC5883L_WRAPPER.cpp](#).

7.16.3.4 `get_type()`

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

Implements [ISensor](#).

Definition at line 39 of file [HMC5883L_WRAPPER.cpp](#).

7.16.3.5 `configure()`

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

Implements [ISensor](#).

Definition at line 43 of file [HMC5883L_WRAPPER.cpp](#).

7.16.4 Member Data Documentation

7.16.4.1 `sensor_`

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

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

7.16.4.2 `initialized_`

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

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

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

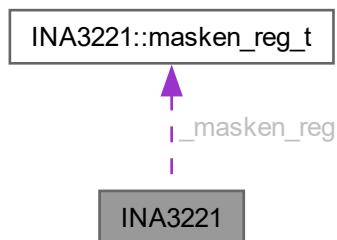
- lib/sensors/HMC5883L/[HMC5883L_WRAPPER.h](#)
- lib/sensors/HMC5883L/[HMC5883L_WRAPPER.cpp](#)

7.17 INA3221 Class Reference

[INA3221](#) Triple-Channel Power Monitor driver class.

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

Collaboration diagram for INA3221:



Classes

- struct `conf_reg_t`
Configuration register bit fields.
- struct `masken_reg_t`
Mask/Enable register bit fields.

Public Member Functions

- `INA3221 (ina3221_addr_t addr, i2c_inst_t *i2c)`
Constructor for `INA3221` class.
- `bool begin ()`
Initialize the `INA3221` device.
- `uint16_t read_register (ina3221_reg_t reg)`
Read a register from the device.
- `void reset ()`
Reset the `INA3221` to default settings.
- `void set_mode_power_down ()`
Set device to power-down mode.
- `void set_mode_continuous ()`
Set device to continuous measurement mode.
- `void set_mode_triggered ()`
Set device to triggered measurement mode.
- `void set_shunt_measurement_enable ()`
Enable shunt voltage measurements.
- `void set_shunt_measurement_disable ()`
Disable shunt voltage measurements.
- `void set_bus_measurement_enable ()`
Enable bus voltage measurements.
- `void set_bus_measurement_disable ()`
Disable bus voltage measurements.
- `void set_averaging_mode (ina3221_avg_mode_t mode)`
Set the averaging mode for measurements.
- `void set_bus_conversion_time (ina3221_conv_time_t convTime)`
Set bus voltage conversion time.
- `void set_shunt_conversion_time (ina3221_conv_time_t convTime)`
Set shunt voltage conversion time.
- `uint16_t get_manufacturer_id ()`
Get the manufacturer ID of the device.
- `uint16_t get_die_id ()`
Get the die ID of the device.
- `int32_t get_shunt_voltage (ina3221_ch_t channel)`
Get shunt voltage for a specific channel.
- `float get_current (ina3221_ch_t channel)`
- `float get_current_ma (ina3221_ch_t channel)`
Get current for a specific channel.
- `float get_voltage (ina3221_ch_t channel)`
Get bus voltage for a specific channel.
- `void set_warn_alert_limit (ina3221_ch_t channel, float voltage_v)`
Set warning alert voltage threshold for a channel.

- void `set_crit_alert_limit` (`ina3221_ch_t` channel, float voltage_v)
Set critical alert voltage threshold for a channel.
- void `set_power_valid_limit` (float voltage_upper_v, float voltage_lower_v)
Set power valid voltage range.
- void `enable_alerts` ()
Enable all alert functions.
- bool `get_warn_alert` (`ina3221_ch_t` channel)
Get warning alert status for a channel.
- bool `get_crit_alert` (`ina3221_ch_t` channel)
Get critical alert status for a channel.
- bool `get_power_valid_alert` ()
Get power valid alert status.
- void `set_alert_latch` (bool enable)
Set alert latch mode.

Private Member Functions

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

Private Attributes

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

7.17.1 Detailed Description

[INA3221](#) Triple-Channel Power Monitor driver class.

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

Definition at line 96 of file [INA3221.h](#).

7.17.2 Member Function Documentation

7.17.2.1 `_read()`

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

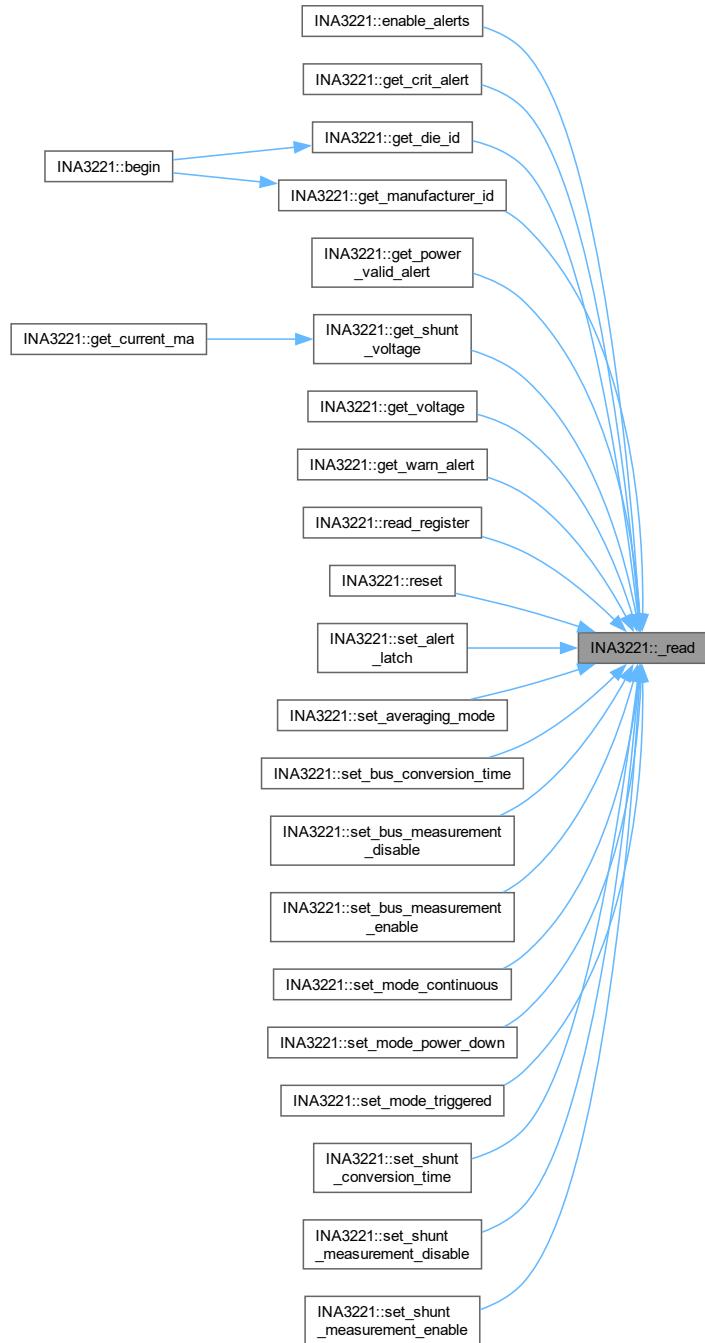
Read a 16-bit register from the device.

Parameters

<i>reg</i>	Register address
<i>val</i>	Pointer to store the read value

Definition at line 513 of file [INA3221.cpp](#).

Here is the caller graph for this function:



7.17.2.2 `_write()`

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

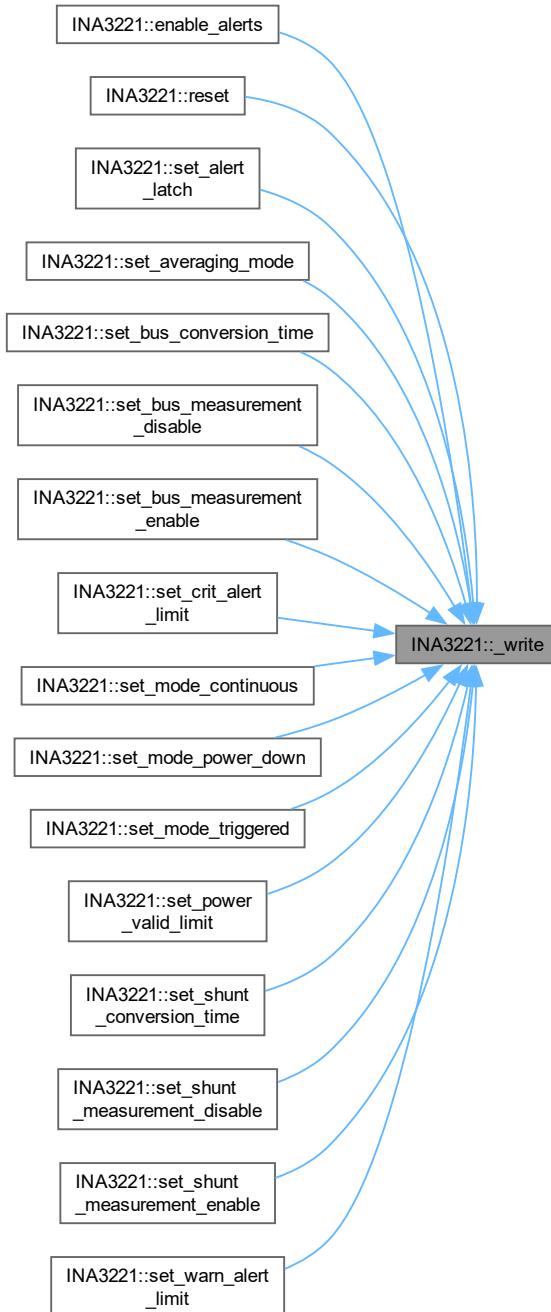
Write a 16-bit value to a register.

Parameters

<i>reg</i>	Register address
<i>val</i>	Pointer to the value to write

Definition at line 539 of file [INA3221.cpp](#).

Here is the caller graph for this function:



7.17.2.3 `get_current()`

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

7.17.3 Member Data Documentation

7.17.3.1 _i2c

```
i2c_inst_t* INA3221::_i2c [private]
```

Definition at line 136 of file [INA3221.h](#).

7.17.3.2 _i2c_addr

```
ina3221_addr_t INA3221::_i2c_addr [private]
```

Definition at line 138 of file [INA3221.h](#).

7.17.3.3 _shuntRes

```
uint32_t INA3221::_shuntRes[INA3221_CH_NUM] [private]
```

Definition at line 141 of file [INA3221.h](#).

7.17.3.4 _filterRes

```
uint32_t INA3221::_filterRes[INA3221_CH_NUM] [private]
```

Definition at line 144 of file [INA3221.h](#).

7.17.3.5 _masken_reg

```
masken_reg_t INA3221::_masken_reg [private]
```

Definition at line 147 of file [INA3221.h](#).

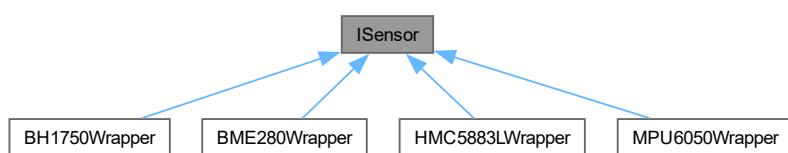
The documentation for this class was generated from the following files:

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

7.18 ISensor Class Reference

```
#include <ISensor.h>
```

Inheritance diagram for ISensor:



Public Member Functions

- virtual `~ISensor ()=default`
- virtual bool `init ()=0`
- virtual float `read_data (SensorDataTypeIdentifier type)=0`
- virtual bool `is_initialized () const =0`
- virtual `SensorType get_type () const =0`
- virtual bool `configure (const std::map< std::string, std::string > &config)=0`

7.18.1 Detailed Description

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

7.18.2 Constructor & Destructor Documentation

7.18.2.1 `~ISensor()`

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

7.18.3 Member Function Documentation

7.18.3.1 `init()`

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

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

7.18.3.2 `read_data()`

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

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

7.18.3.3 `is_initialized()`

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

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

7.18.3.4 `get_type()`

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

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

7.18.3.5 `configure()`

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

Implemented in [BH1750Wrapper](#), [BME280Wrapper](#), [HMC5883LWrapper](#), and [MPU6050Wrapper](#).

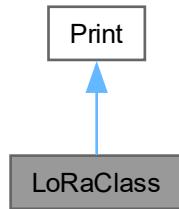
The documentation for this class was generated from the following file:

- lib/sensors/ISensor.h

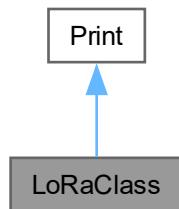
7.19 LoRaClass Class Reference

```
#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](#).

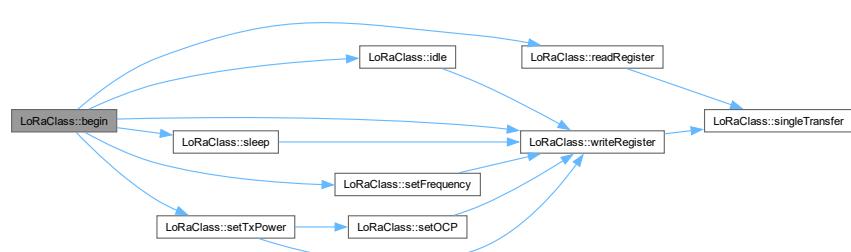
7.19.3 Member Function Documentation

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:

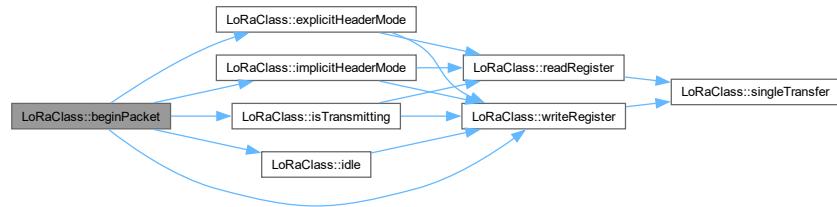


7.19.3.3 beginPacket()

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

Definition at line 158 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

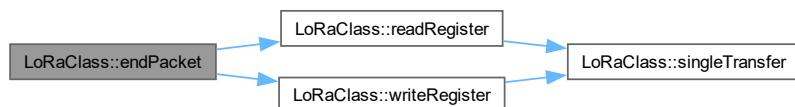


7.19.3.4 endPacket()

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

Definition at line 180 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

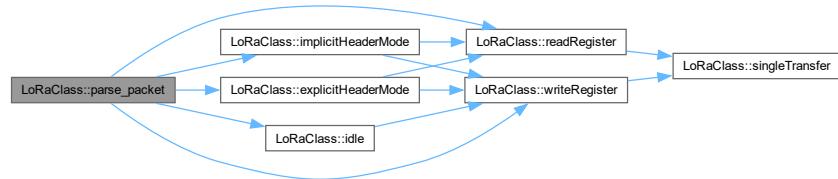


7.19.3.5 parse_packet()

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

Definition at line 215 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

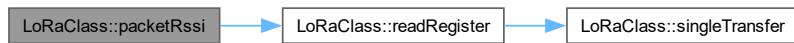


7.19.3.6 packetRssi()

```
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:



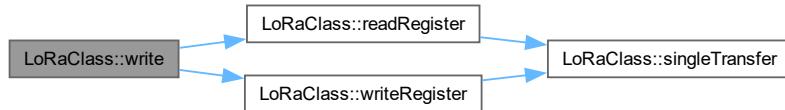
7.19.3.11 write() [2/2]

```
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()

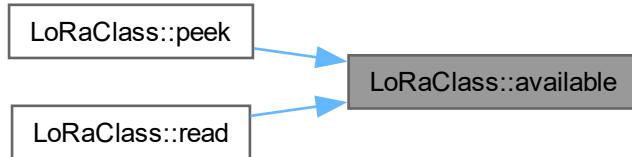
```
int LoRaClass::available ()  [virtual]
```

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:

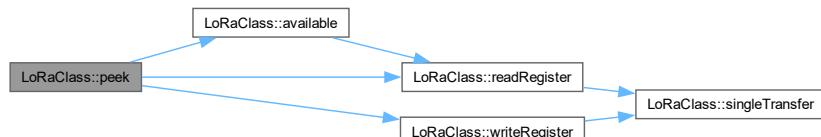


7.19.3.14 `peek()`

```
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:

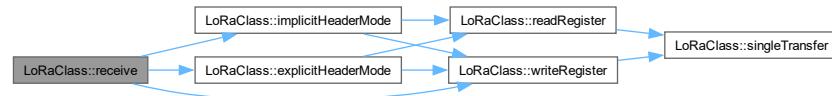


7.19.3.19 receive()

```
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()

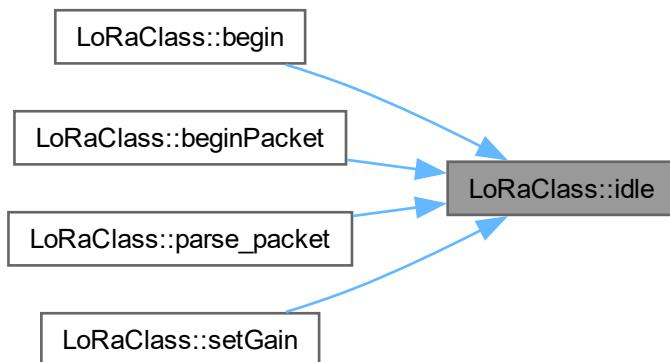
```
void LoRaClass::idle ()
```

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()

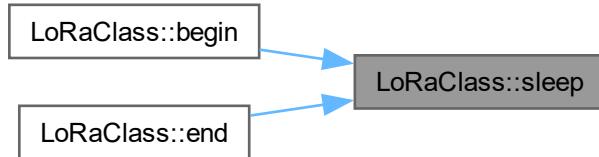
```
void LoRaClass::sleep ()
```

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:

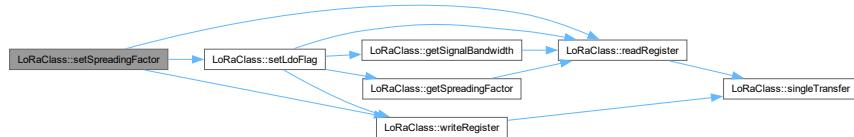


7.19.3.25 setSpreadingFactor()

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

Definition at line 477 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

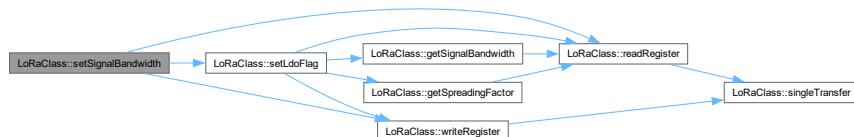


7.19.3.26 setSignalBandwidth()

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

Definition at line 517 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

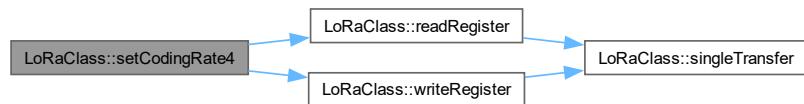


7.19.3.27 setCodingRate4()

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

Definition at line 560 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

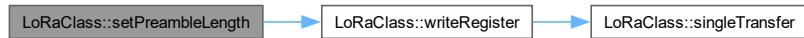


7.19.3.28 setPreambleLength()

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

Definition at line 573 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

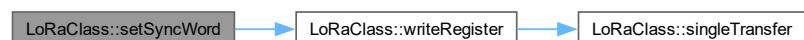


7.19.3.29 setSyncWord()

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

Definition at line 579 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

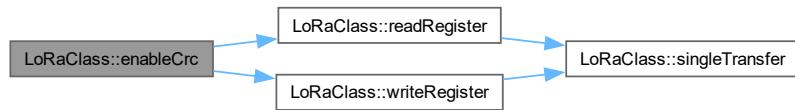


7.19.3.30 enableCrc()

```
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:

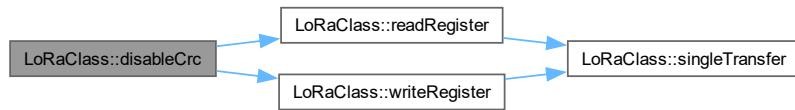


7.19.3.31 disableCrc()

```
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()

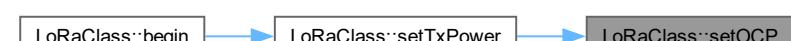
```
void LoRaClass::setOCP (
    uint8_t mA)
```

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:

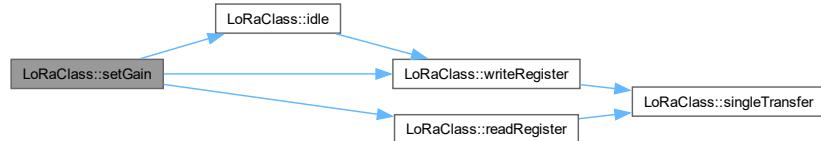


7.19.3.35 setGain()

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

Definition at line 619 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

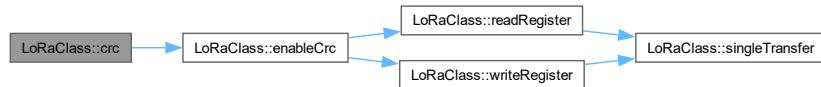


7.19.3.36 crc()

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

Definition at line 73 of file [LoRa-RP2040.h](#).

Here is the call graph for this function:

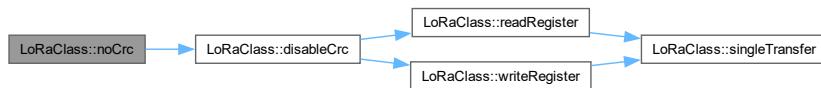


7.19.3.37 noCrc()

```
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:

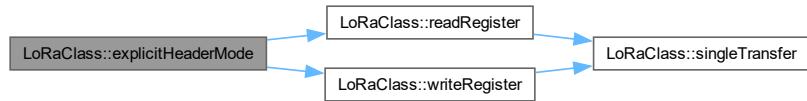


7.19.3.43 explicitHeaderMode()

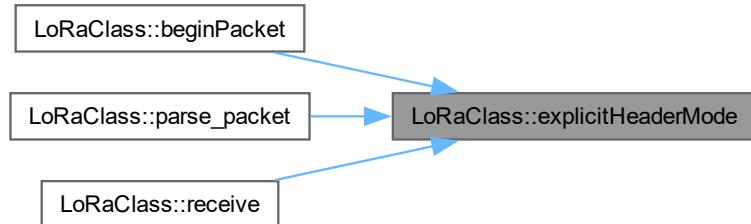
```
void LoRaClass::explicitHeaderMode () [private]
```

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:

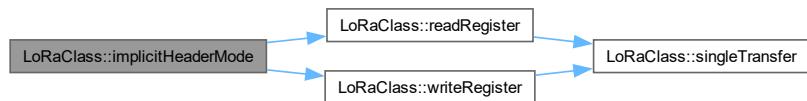


7.19.3.44 implicitHeaderMode()

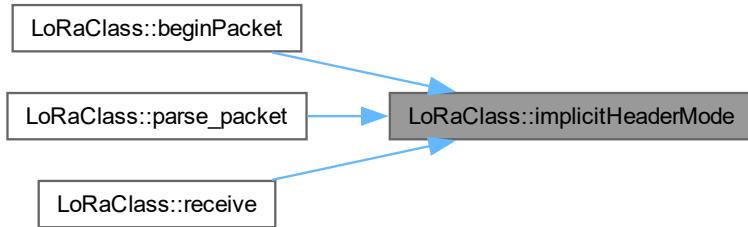
```
void LoRaClass::implicitHeaderMode () [private]
```

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:

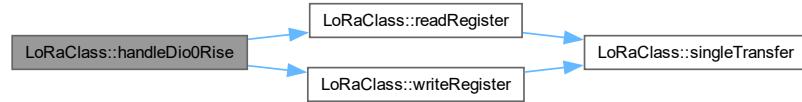


7.19.3.45 handleDio0Rise()

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

Definition at line 688 of file [LoRa-RP2040.cpp](#).

Here is the call graph for this function:

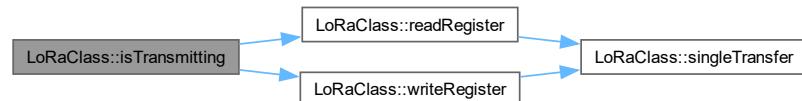


7.19.3.46 isTransmitting()

```
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:

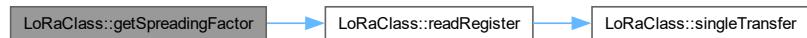


7.19.3.47 getSpreadingFactor()

```
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:

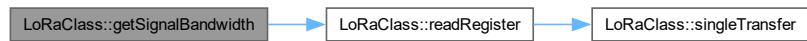


7.19.3.48 getSignalBandwidth()

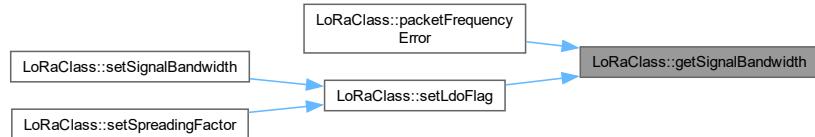
```
long LoRaClass::getSignalBandwidth () [private]
```

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:

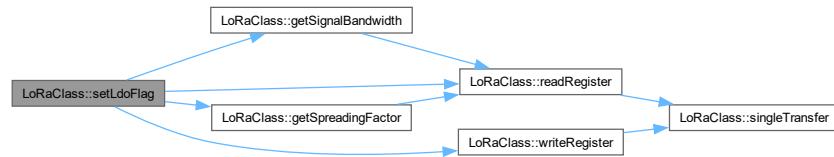


7.19.3.49 setLdoFlag()

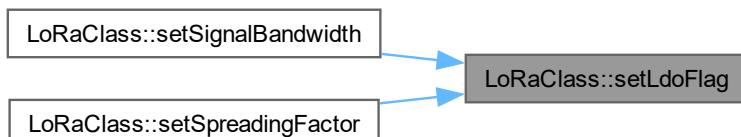
```
void LoRaClass::setLdoFlag () [private]
```

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()

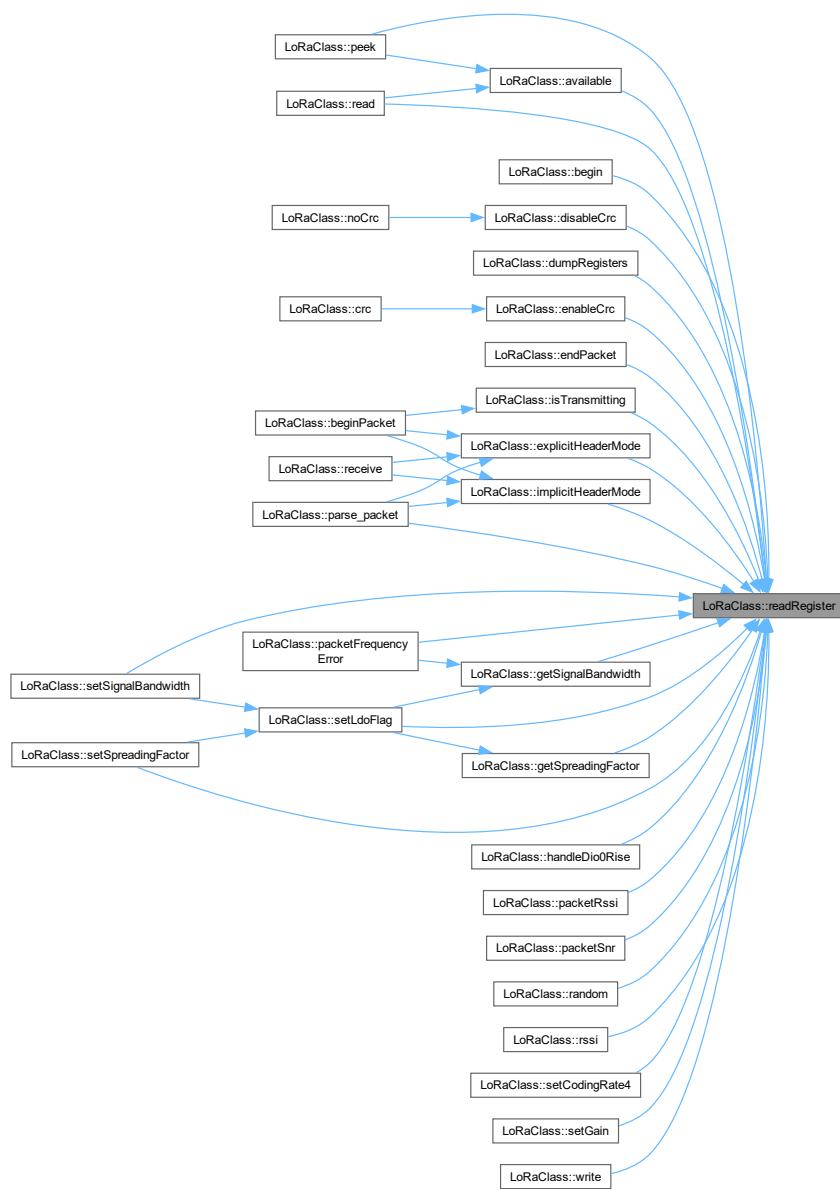
```
uint8_t LoRaClass::readRegister (
    uint8_t address) [private]
```

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()

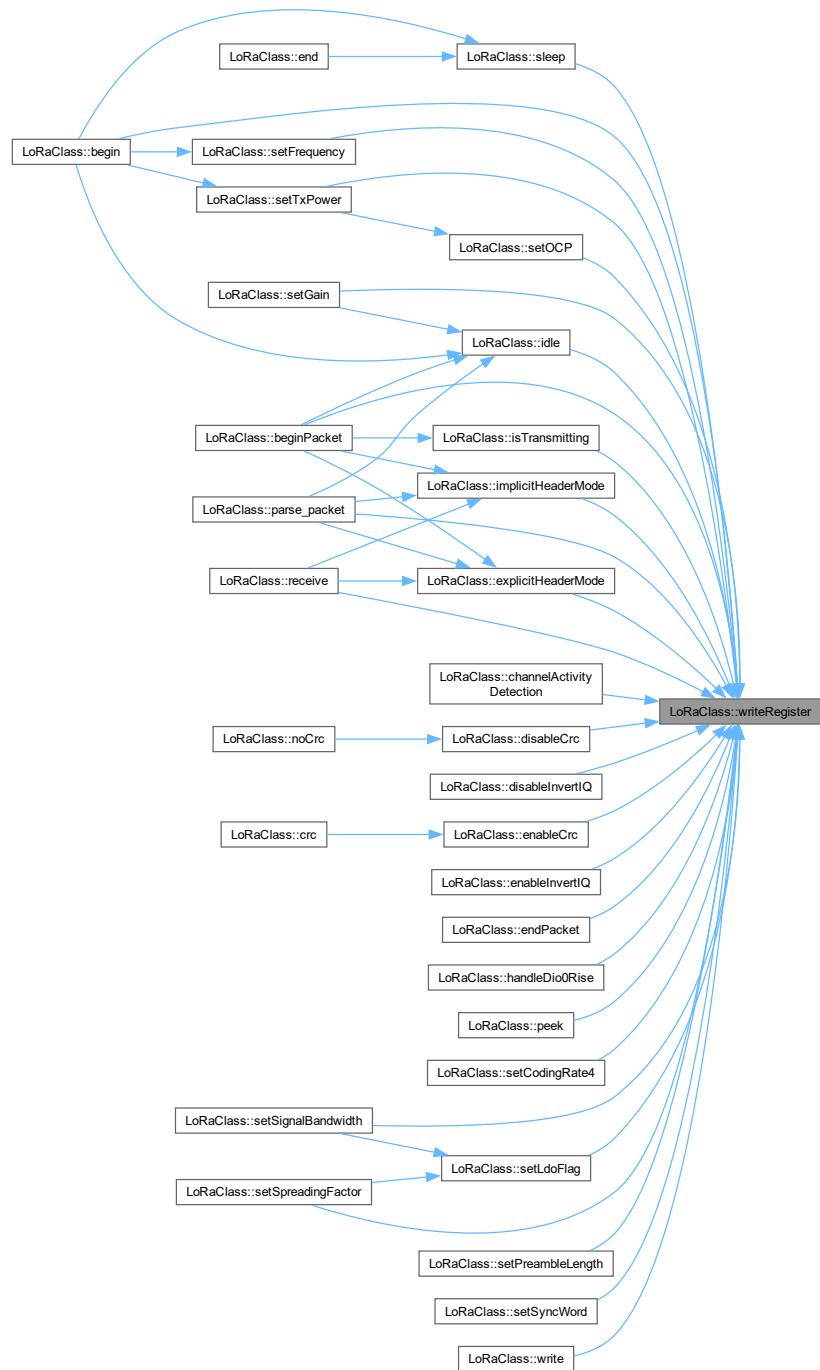
```
void LoRaClass::writeRegister (
    uint8_t address,
    uint8_t value) [private]
```

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:



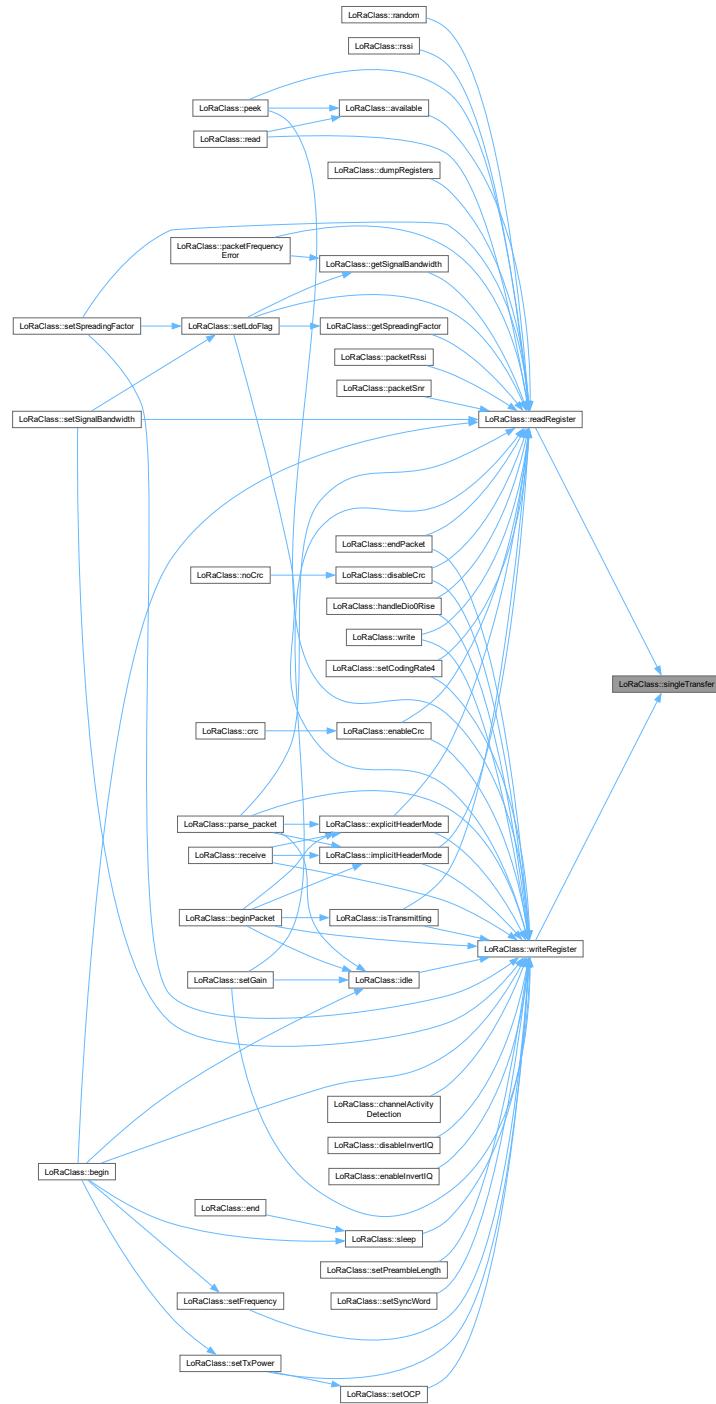
7.19.3.52 `singleTransfer()`

```

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:

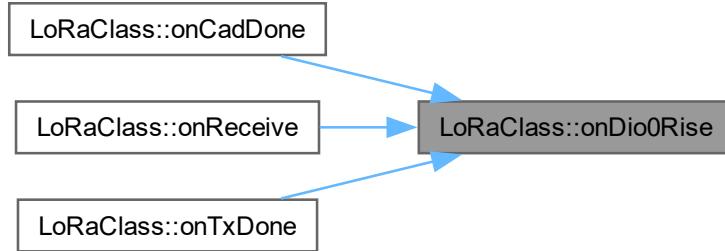


7.19.3.53 onDio0Rise()

```
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](#).

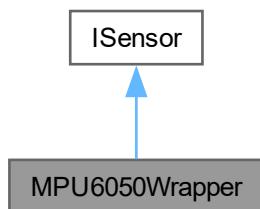
The documentation for this struct was generated from the following file:

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

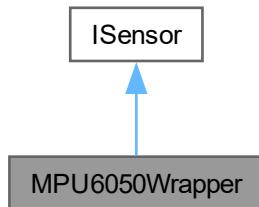
7.21 MPU6050Wrapper Class Reference

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

Inheritance diagram for MPU6050Wrapper:



Collaboration diagram for MPU6050Wrapper:



Public Member Functions

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

Public Member Functions inherited from [ISensor](#)

- `virtual ~ISensor ()=default`

Private Attributes

- `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](#).

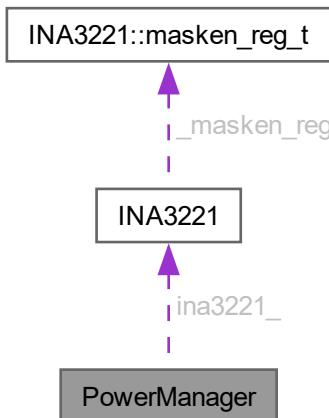
The documentation for this class was generated from the following files:

- 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 ()
```

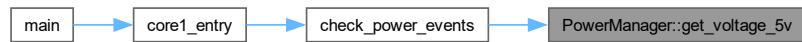
Definition at line 36 of file [PowerManager.cpp](#).

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()`

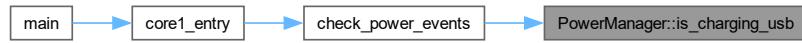
```
bool PowerManager::is_charging_usb ()
```

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](#).

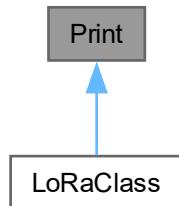
The documentation for this class was generated from the following files:

- 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()

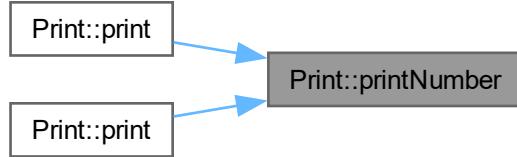
```
size_t Print::printNumber (
    unsigned long n,
    uint8_t base) [private]
```

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()

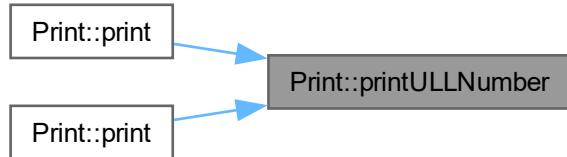
```
size_t Print::printULLNumber (
    unsigned long long n64,
    uint8_t base) [private]
```

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:

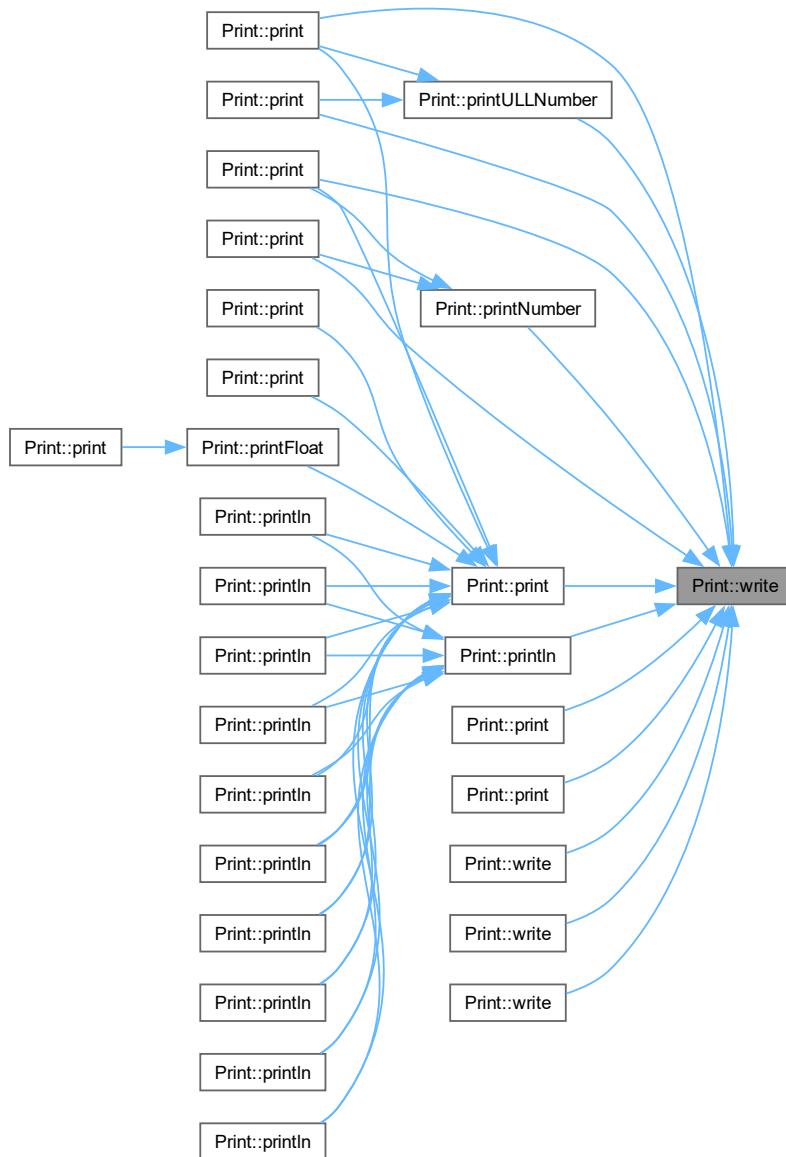


7.24.3.7 write() [1/4]

```
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]

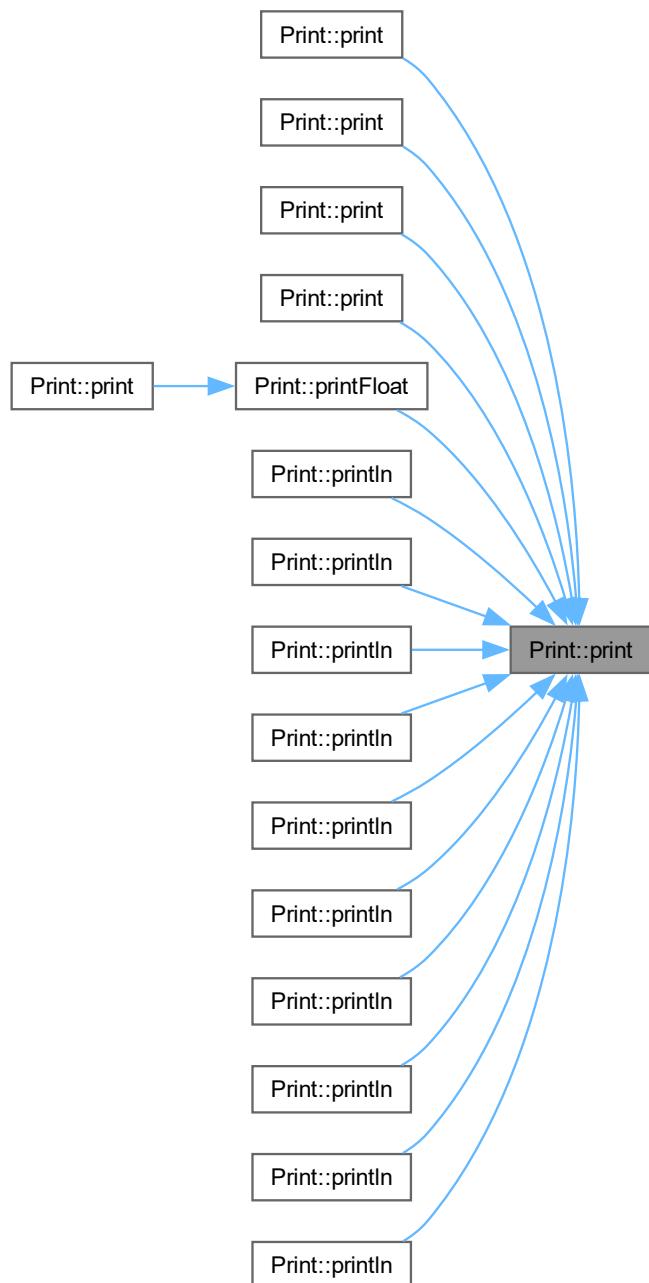
```
size_t Print::print (
    char c)
```

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:

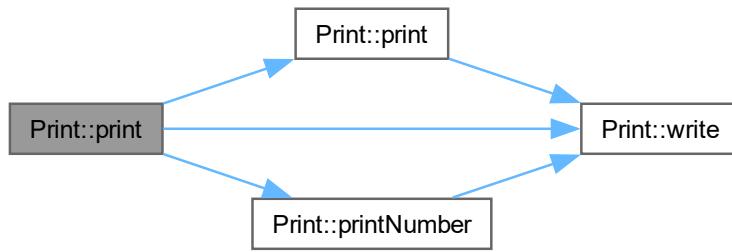


7.24.3.18 print() [7/11]

```
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:

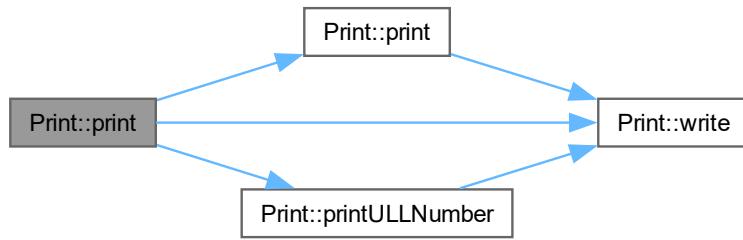


7.24.3.20 print() [9/11]

```
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](#).

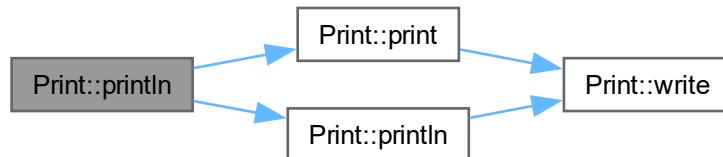
Here is the call graph for this function:

**7.24.3.23 println() [1/11]**

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

Definition at line 130 of file [Print.cpp](#).

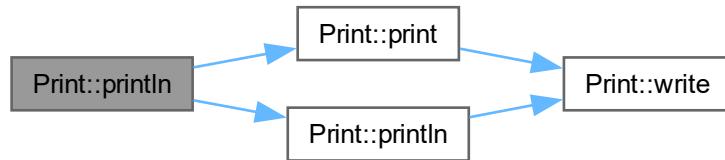
Here is the call graph for this function:

**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:

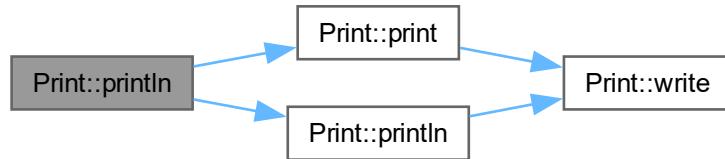


7.24.3.25 `println()` [3/11]

```
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:

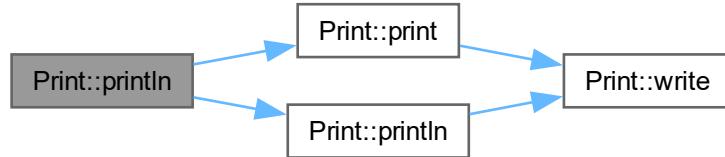


7.24.3.26 `println()` [4/11]

```
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:

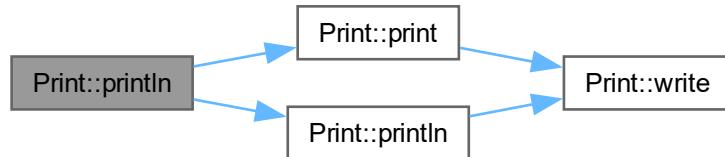


7.24.3.27 `println()` [5/11]

```
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:

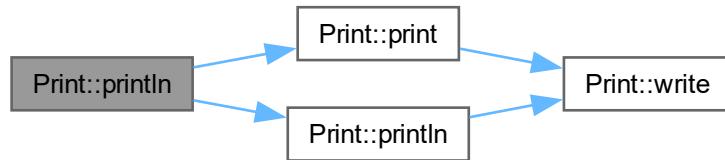


7.24.3.28 `println()` [6/11]

```
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:

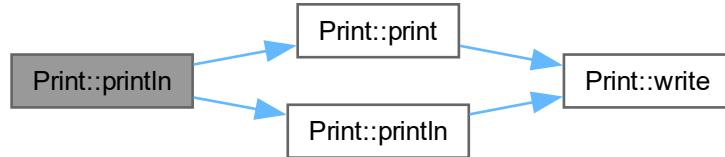


7.24.3.29 println() [7/11]

```
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:

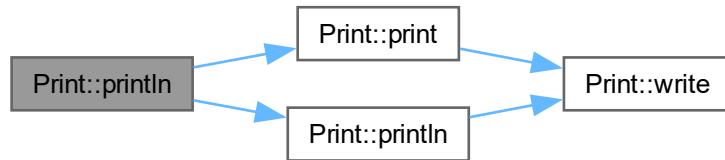


7.24.3.30 println() [8/11]

```
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:

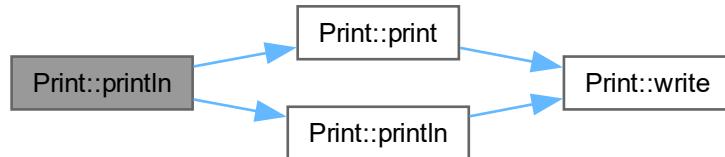


7.24.3.31 `println()` [9/11]

```
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:

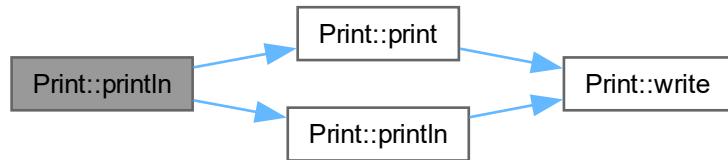


7.24.3.32 `println()` [10/11]

```
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]

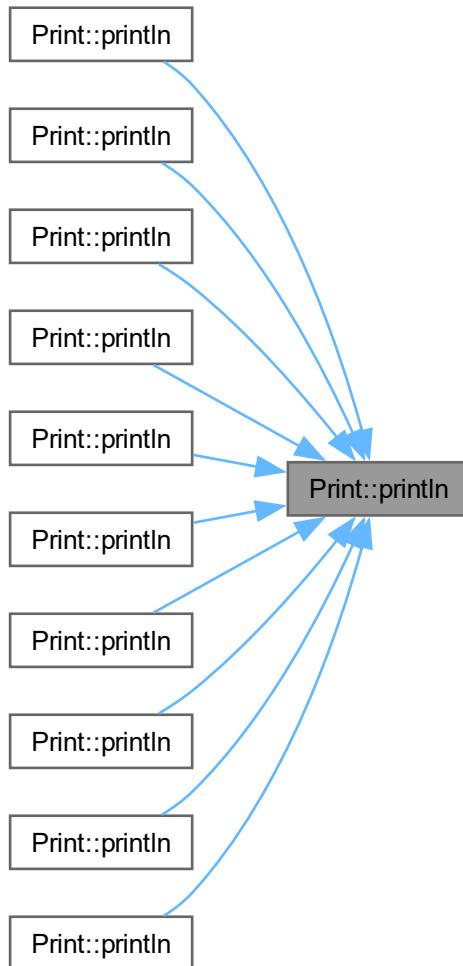
```
size_t Print::println (
    void )
```

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, SensorDataTypelIdentifier dataType)`
Reads a specific data type (e.g., temperature, humidity) from a sensor.

Static Public Member Functions

- static `SensorWrapper & get_instance ()`
Provides a global instance of SensorWrapper.

Private Member Functions

- `SensorWrapper ()`
Default constructor for SensorWrapper.

Private Attributes

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

7.25.1 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

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

Returns

True if initialization succeeded, otherwise false.

Definition at line 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

<i>type</i>	The sensor type.
<i>config</i>	Key-value pairs for sensor configuration.

Returns

True if the sensor was successfully configured, otherwise false.

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

7.25.3.4 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

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

Returns

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

Definition at line 78 of file [ISensor.cpp](#).

7.25.4 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:

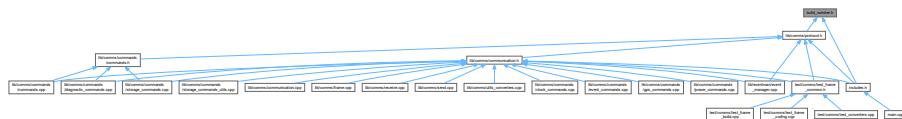
- lib/sensors/[ISensor.h](#)
- lib/sensors/[ISensor.cpp](#)

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 343`

8.1.1 Macro Definition Documentation

8.1.1.1 BUILD_NUMBER

```
#define BUILD_NUMBER 343
```

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 343
00007
00008 #endif
```

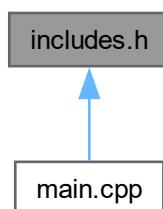
8.3 includes.h File Reference

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

Include dependency graph for includes.h:



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



8.4 includes.h

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

```

00001 #ifndef INCLUDES_H
00002 #define INCLUDES_H
00003
00004 #include <stdio.h>
00005 #include "pico/stl.h"
00006 #include "hardware/spi.h"
00007 #include "hardware/i2c.h"
00008 #include "hardware/uart.h"
00009 #include "pico/multicore.h"
00010 #include "event_manager.h"
00011 #include "lib/powerman/PowerManager.h" // Corrected path
00012
00013 #include "ISensor.h"
00014 #include "lib/sensors/BH1750/BH1750_WRAPPER.h" // Corrected path
00015 #include "lib/sensors/BME280/BME280_WRAPPER.h" // Corrected path
00016 #include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h" // Corrected path
00017 #include "lib/sensors/MPU6050/MPU6050_WRAPPER.h" // Corrected path
00018 #include "lib/clock/DS3231.h" // Corrected path
00019 #include <iostream>
00020 #include <iomanip>
00021 #include <queue>
00022 #include <chrono>
00023 #include "protocol.h"
00024 #include <atomic>
00025 #include <iostream>
00026 #include <map>
00027 #include "pin_config.h"
00028 #include "utils.h"
00029 #include "communication.h"
00030 #include "build_number.h"
00031 #include "lib/location/gps_collector.h"
00032 #include "lib/storage/storage.h" // Corrected path
00033 #include "lib/storage/pico-vfs/include/filesystem/vfs.h" // Corrected path
00034
00035 #endif

```

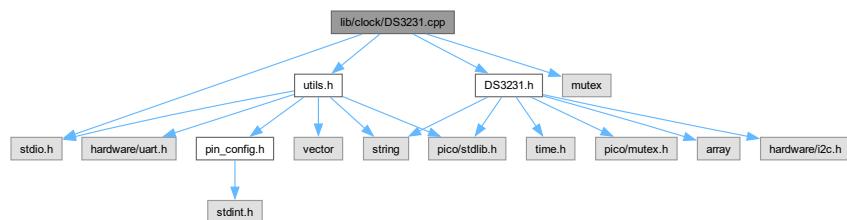
8.5 lib/clock/DS3231.cpp File Reference

```

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

```

Include dependency graph for DS3231.cpp:



8.6 DS3231.cpp

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

```

00001 #include "DS3231.h"
00002 #include "utils.h"
00003 #include <stdio.h> // Include for printf

```

```

00004 #include <mutex> // Include for mutex
00005
00006 DS3231::DS3231(i2c_inst_t *i2c_instance) : i2c(i2c_instance), ds3231_addr(DS3231_DEVICE_ADDRESS) {
00007     // Initialize mutex (assuming you have a mutex member variable)
00008     recursive_mutex_init(&clock_mutex_);
00009 }
0010
0011 int DS3231::set_time(ds3231_data_t *data) {
0012     uint8_t temp[7] = {0};
0013
0014     // Enable oscillator
0015     if (clock_enable() != 0) {
0016         uart_print("Failed to enable clock oscillator", VerbosityLevel::ERROR);
0017         return -1;
0018     }
0019
0020     if (data->seconds > 59)
0021         data->seconds = 59;
0022     if (data->minutes > 59)
0023         data->minutes = 59;
0024     if (data->hours > 23)
0025         data->hours = 23;
0026     if (data->day > 7)
0027         data->day = 7;
0028     else if (data->day < 1)
0029         data->day = 1;
0030     if (data->date > 31)
0031         data->date = 31;
0032     else if (data->date < 1)
0033         data->date = 1;
0034     if (data->month > 12)
0035         data->month = 12;
0036     else if (data->month < 1)
0037         data->month = 1;
0038     if (data->year > 99)
0039         data->year = 99;
0040
0041     temp[0] = bin_to_bcd(data->seconds);
0042     temp[1] = bin_to_bcd(data->minutes);
0043     temp[2] = bin_to_bcd(data->hours);
0044     temp[2] &= ~(0x01 « 6); // Clear 12/24 hour bit
0045     temp[3] = bin_to_bcd(data->day);
0046     temp[4] = bin_to_bcd(data->date);
0047     temp[5] = bin_to_bcd(data->month);
0048     if (data->century)
0049         temp[5] |= (0x01 « 7);
0050     temp[6] = bin_to_bcd(data->year);
0051
0052     std::string status = "BCD values to be written to DS3231: " + std::to_string(temp[0]) + " " +
0053             std::to_string(temp[1]) + " " + std::to_string(temp[2]) + " " +
0054             std::to_string(temp[3]) + " " + std::to_string(temp[4]) + " " +
0055             std::to_string(temp[5]) + " " + std::to_string(temp[6]);
0056
0057     uart_print(status, VerbosityLevel::DEBUG);
0058
0059     int result = i2c_write_reg(DS3231_SECONDS_REG, 7, temp);
0060     if (result != 0) {
0061         uart_print("i2c write failed", VerbosityLevel::ERROR);
0062         return -1;
0063     }
0064
0065     return 0;
0066 }
0067
0068 int DS3231::get_time(ds3231_data_t *data) {
0069     std::string status;
0070     uint8_t raw_data[7];
0071     int result = i2c_read_reg(DS3231_SECONDS_REG, 7, raw_data);
0072     if (result != 0) {
0073         status = "Failed to read time from DS3231";
0074         uart_print(status, VerbosityLevel::ERROR);
0075         return -1;
0076     }
0077
0078     status = "Raw BCD values read from DS3231: " + std::to_string(raw_data[0]) + " " +
0079             std::to_string(raw_data[1]) + " " + std::to_string(raw_data[2]) + " " +
0080             std::to_string(raw_data[3]) + " " + std::to_string(raw_data[4]) + " " +
0081             std::to_string(raw_data[5]) + " " + std::to_string(raw_data[6]);
0082     uart_print(status, VerbosityLevel::DEBUG);
0083
0084     data->seconds = bcd_to_bin(raw_data[0] & 0x7F); // Masking for CH bit (clock halt)
0085     data->minutes = bcd_to_bin(raw_data[1] & 0x7F);
0086     data->hours = bcd_to_bin(raw_data[2] & 0x3F); // Masking for 12/24 hour mode bit
0087     data->day = raw_data[3] & 0x07; // Day of week (1-7)
0088     data->date = bcd_to_bin(raw_data[4] & 0x3F);
0089     data->month = bcd_to_bin(raw_data[5] & 0x1F); // Masking for century bit
0090     data->century = (raw_data[5] & 0x80) » 7;

```

```

00091     data->year = bcd_to_bin(raw_data[6]);
00092
00093     // Data validation
00094     if (data->seconds > 59 || data->minutes > 59 || data->hours > 23 ||
00095         data->day < 1 || data->day > 7 || data->date < 1 || data->date > 31 ||
00096         data->month < 1 || data->month > 12 || data->year > 99) {
00097         uart_print("Invalid data read from DS3231", VerbosityLevel::ERROR);
00098         return -1;
00099     }
00100
00101     uart_print("Reading time from DS3231", VerbosityLevel::DEBUG);
00102     std::string timeStr = "Time: " + std::to_string(data->hours) + ":" + std::to_string(data->minutes)
00103     + ":" + std::to_string(data->seconds);
00104     uart_print(timeStr, VerbosityLevel::DEBUG);
00105     std::string dateStr = "Date: " + std::to_string(data->date) + "/" + std::to_string(data->month) +
00106     "/" + std::to_string(data->year);
00107     uart_print(dateStr, VerbosityLevel::DEBUG);
00108
00109     return 0;
00110 }
00111
00112 int DS3231::read_temperature(float *resolution) {
00113     std::string status;
00114     uint8_t temp[2];
00115     int result = i2c_read_reg(DS3231_TEMPERATURE_MSB_REG, 2, temp);
00116     if (result != 0) {
00117         status = "Failed to read temperature from DS3231";
00118         uart_print(status, VerbosityLevel::ERROR);
00119         return -1;
00120     }
00121
00122     int8_t temperature_msb = (int8_t)temp[0]; // Signed for negative temperatures
00123     uint8_t temperature_lsb = temp[1] >> 6; // Only the 2 MSB are valid
00124
00125     *resolution = temperature_msb + (temperature_lsb * 0.25f); // 0.25 degree resolution
00126
00127     return 0;
00128 }
00129
00130 int DS3231::i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
00131     if (!length)
00132         return -1;
00133
00134     std::string status = "Reading register " + std::to_string(reg_addr) + " from DS3231";
00135     uart_print(status, VerbosityLevel::DEBUG);
00136     recursive_mutex_enter_blocking(&clock_mutex_);
00137     uint8_t reg = reg_addr;
00138     int write_result = i2c_write_blocking(i2c, ds3231_addr, &reg, 1, true);
00139     if (write_result == PICO_ERROR_GENERIC) {
00140         status = "Failed to write register address to DS3231";
00141         uart_print(status, VerbosityLevel::ERROR);
00142         recursive_mutex_exit(&clock_mutex_);
00143         return -1;
00144     }
00145     int read_result = i2c_read_blocking(i2c, ds3231_addr, data, length, false);
00146     if (read_result == PICO_ERROR_GENERIC) {
00147         status = "Failed to read register data from DS3231";
00148         uart_print(status, VerbosityLevel::ERROR);
00149         recursive_mutex_exit(&clock_mutex_);
00150         return -1;
00151     }
00152     recursive_mutex_exit(&clock_mutex_);
00153
00154     return 0;
00155 }
00156
00157
00158 recursive_mutex_exit(&clock_mutex_);
00159
00160
00161 }
00162
00163 int DS3231::i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data) {
00164     if (!length)
00165         return -1;
00166
00167     recursive_mutex_enter_blocking(&clock_mutex_);
00168     uint8_t message[length + 1];
00169     message[0] = reg_addr;
00170     for (int i = 0; i < length; i++) {
00171         message[i + 1] = data[i];
00172     }
00173     int write_result = i2c_write_blocking(i2c, ds3231_addr, message, (length + 1), false);
00174     if (write_result == PICO_ERROR_GENERIC) {
00175         uart_print("Error: i2c_write_blocking failed in i2c_write_reg", VerbosityLevel::ERROR);
00176         recursive_mutex_exit(&clock_mutex_);
00177         return -1;
00178     }
00179     recursive_mutex_exit(&clock_mutex_);
00180
00181     return 0;
00182 }
00183
00184 }
```

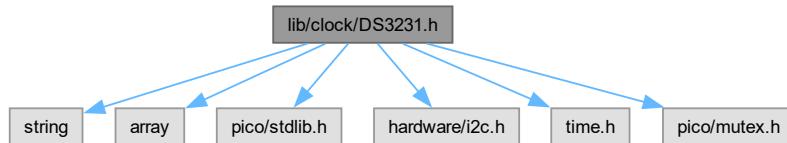
```

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

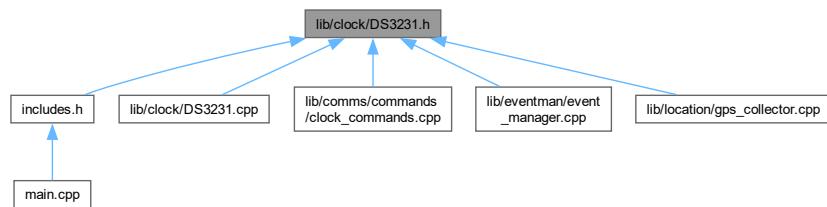
```

8.7 lib/clock/DS3231.h File Reference

```
#include <string>
#include <array>
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include <time.h>
#include "pico/mutex.h"
Include dependency graph for DS3231.h:
```



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



Classes

- struct `ds3231_data_t`
- class `DS3231`

Macros

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

Enumerations

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

8.7.1 Macro Definition Documentation

8.7.1.1 DS3231_DEVICE_ADDRESS

```
#define DS3231_DEVICE_ADDRESS 0x68
```

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

8.7.1.2 DS3231_SECONDS_REG

```
#define DS3231_SECONDS_REG 0x00
```

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

8.7.1.3 DS3231_MINUTES_REG

```
#define DS3231_MINUTES_REG 0x01
```

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

8.7.1.4 DS3231_HOURS_REG

```
#define DS3231_HOURS_REG 0x02
```

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

8.7.1.5 DS3231_DAY_REG

```
#define DS3231_DAY_REG 0x03
```

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

8.7.1.6 DS3231_DATE_REG

```
#define DS3231_DATE_REG 0x04
```

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

8.7.1.7 DS3231_MONTH_REG

```
#define DS3231_MONTH_REG 0x05
```

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

8.7.1.8 DS3231_YEAR_REG

```
#define DS3231_YEAR_REG 0x06
```

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

8.7.1.9 DS3231_CONTROL_REG

```
#define DS3231_CONTROL_REG 0x0E
```

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

8.7.1.10 DS3231_CONTROL_STATUS_REG

```
#define DS3231_CONTROL_STATUS_REG 0x0F
```

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

8.7.1.11 DS3231_TEMPERATURE_MSB_REG

```
#define DS3231_TEMPERATURE_MSB_REG 0x11
```

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

8.7.1.12 DS3231_TEMPERATURE_LSB_REG

```
#define DS3231_TEMPERATURE_LSB_REG 0x12
```

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

8.7.2 Enumeration Type Documentation

8.7.2.1 days_of_week

```
enum days_of_week
```

Enumerator

MONDAY	
TUESDAY	
WEDNESDAY	
THURSDAY	
FRIDAY	
SATURDAY	
SUNDAY	

Definition at line 27 of file [DS3231.h](#).

8.8 DS3231.h

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

```

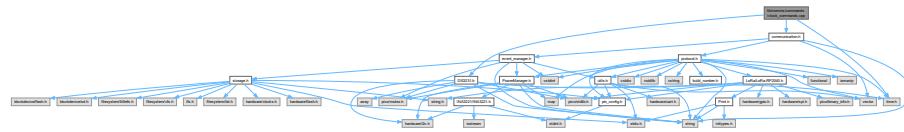
00001 #ifndef DS3231_H
00002 #define DS3231_H
00003
00004 #include <string>
00005 #include <array>
00006 #include "pico/stdlib.h"
00007 #include "hardware/i2c.h"
00008 #include <time.h>
00009 #include "pico/mutex.h"
00010
00011 #define DS3231_DEVICE_ADDRESS 0x68
00012
00013 #define DS3231_SECONDS_REG 0x00
00014 #define DS3231_MINUTES_REG 0x01
00015 #define DS3231_HOURS_REG 0x02
00016 #define DS3231_DAY_REG 0x03
00017 #define DS3231_DATE_REG 0x04
00018 #define DS3231_MONTH_REG 0x05
00019 #define DS3231_YEAR_REG 0x06
00020
00021 #define DS3231_CONTROL_REG 0x0E
00022 #define DS3231_CONTROL_STATUS_REG 0x0F
00023
00024 #define DS3231_TEMPERATURE_MSB_REG 0x11
00025 #define DS3231_TEMPERATURE_LSB_REG 0x12
00026
00027 enum days_of_week {
00028     MONDAY = 1,
00029     TUESDAY,
00030     WEDNESDAY,
00031     THURSDAY,
00032     FRIDAY,
00033     SATURDAY,
00034     SUNDAY
00035 };
00036
00037 typedef struct {
00038     uint8_t seconds;
00039     uint8_t minutes;
00040     uint8_t hours;
00041     uint8_t day;
00042     uint8_t date;
00043     uint8_t month;
00044     uint8_t year;
00045     bool century;
00046 } ds3231_data_t;
00047
00048 class DS3231 {
00049 public:
00050     DS3231(i2c_inst_t *i2c_instance);
00051
00052     int set_time(ds3231_data_t *data);
00053     int get_time(ds3231_data_t *data);
00054     int read_temperature(float *resolution);
00055
00056     int set_unix_time(time_t unix_time);
00057     time_t get_unix_time();
00058     int clock_enable();
00059
00060 private:
00061     i2c_inst_t *i2c;
00062     uint8_t ds3231_addr;
00063
00064     int i2c_read_reg(uint8_t reg_addr, size_t length, uint8_t *data);
00065     int i2c_write_reg(uint8_t reg_addr, size_t length, uint8_t *data);
00066
00067     uint8_t bin_to_bcd(const uint8_t data);
00068     uint8_t bcd_to_bin(const uint8_t bcd);
00069
00070     recursive_mutex_t clock_mutex; // Mutex for I2C access
00071 };
00072
00073 #endif

```

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

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

```
#include "DS3231.h"
Include dependency graph for clock_commands.cpp:
```



Macros

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

Functions

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

Variables

- DS3231 systemClock

8.9.1 Macro Definition Documentation

8.9.1.1 CLOCK_GROUP

```
#define CLOCK_GROUP 3
```

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

8.9.1.2 TIME

```
#define TIME 0
```

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

8.9.1.3 TIMEZONE_OFFSET

```
#define TIMEZONE_OFFSET 1
```

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

8.9.1.4 CLOCK_SYNC_INTERVAL

```
#define CLOCK_SYNC_INTERVAL 2
```

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

8.9.1.5 LAST_SYNC_TIME

```
#define LAST_SYNC_TIME 3
```

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

8.10 `clock_commands.cpp`

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

```
00001 #include "communication.h"
00002 #include <time.h>
00003 #include "DS3231.h" // Include the DS3231 header
00004
00005 #define CLOCK_GROUP 3
00006 #define TIME 0
00007 #define TIMEZONE_OFFSET 1
00008 #define CLOCK_SYNC_INTERVAL 2
00009 #define LAST_SYNC_TIME 3
00010
00016
00017 extern DS3231 systemClock;
00018
00032 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             if (systemClock.set_unix_time(newTime) != 0) return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "FAILED TO SET TIME");
00041             EventEmitter::emit(EventGroup::CLOCK, ClockEvent::CHANGED);
00042             return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, TIME, std::to_string(systemClock.get_unix_time()));
00044         } 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         uint32_t timeUnix = systemClock.get_unix_time();
00052         if (timeUnix == 0) return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIME, "FAILED TO GET TIME");
00053         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 }
```

```
00059
00060
00071 Frame handle_timezone_offset(const std::string& param, OperationType operationType) {
00072     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00073         return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID OPERATION");
00074     }
00075
00076     if (operationType == OperationType::GET) {
00077         if (!param.empty()) {
00078             return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "PARAM
UNNECESSARY");
00079         }
00080
00081         std::string timezoneOffset = "60";
00082         return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, TIMEZONE_OFFSET, timezoneOffset);
00083     }
00084
00085     if (operationType == OperationType::SET) {
00086         if (param.empty()) {
00087             return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "PARAM
REQUIRED");
00088         }
00089         try {
00090             int16_t offset = std::stoi(param);
00091             if (offset < -720 || offset > 720) { // ±12 hours in minutes
00092                 return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID
OFFSET");
00093             }
00094
00095             // set offset
00096             return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, TIMEZONE_OFFSET, param);
00097         } catch (...) {
00098             return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "INVALID
PARAMETER");
00099         }
00100     }
00101     return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, TIMEZONE_OFFSET, "UNKNOWN ERROR");
00102 }
00103
00104
00115 Frame handle_clock_sync_interval(const std::string& param, OperationType operationType) {
00116     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00117         return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "INVALID
OPERATION");
00118     }
00119
00120     if (operationType == OperationType::GET) {
00121         if (!param.empty()) {
00122             return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "PARAM
UNNECESSARY");
00123         }
00124
00125         std::string clockSyncInterval = "1440";
00126         return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, CLOCK_SYNC_INTERVAL,
clockSyncInterval);
00127     }
00128
00129     if (operationType == OperationType::SET) {
00130         if (param.empty()) {
00131             return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "PARAM
REQUIRED");
00132         }
00133         try {
00134             uint32_t interval = std::stoul(param);
00135
00136             //set sync interval
00137
00138             return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, param);
00139         } catch (...) {
00140             return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "INVALID
PARAMETER");
00141         }
00142     }
00143     return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, CLOCK_SYNC_INTERVAL, "UNKNOWN ERROR");
00144 }
00145
00155 Frame handle_get_last_sync_time(const std::string& param, OperationType operationType) {
00156     if (operationType != OperationType::GET || !param.empty()) {
00157         return frame_build(ExecutionResult::ERROR, CLOCK_GROUP, LAST_SYNC_TIME, "INVALID REQUEST");
00158     }
00159     std::string lastSyncTime = "none";
00160     return frame_build(ExecutionResult::SUCCESS, CLOCK_GROUP, LAST_SYNC_TIME, lastSyncTime);
00161 } // 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 ¶m, OperationType operationType)
Executes a command based on its key.

Variables

- **CommandMap commandHandlers**
Global map of all command handlers.

8.12 commands.cpp

Go to the documentation of this file.

```
00001 // commands/commands.cpp
00002 #include "commands.h"
00003 #include "communication.h"
00004
00010
00015 using CommandHandler = std::function<Frame(const std::string&, OperationType)>;
00016
00021 using CommandMap = std::map<uint32_t, CommandHandler>;
00022
00027 CommandMap commandHandlers = {
00028     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(0)), handle_get_commands_list},
    // Group 1, Command 0
00029     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(1)), handle_get_build_version},
    // Group 1, Command 1
00030     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(8)), handle_verbosity},
    // Group 1, Command 9
00031     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(9)), handle_enter_bootloader_mode},
    // Group 1, Command 9
00032     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(0)), handle_get_power_manager_ids},
    // Group 2, Command 0
00033     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(2)), handle_get_voltage_battery},
    // Group 2, Command 2
00034     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(3)), handle_get_voltage_5v},
    // Group 2, Command 3
00035     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(4)), handle_get_current_charge_usb},
    // Group 2, Command 4
```

```

00036     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(5)), handle_get_current_charge_solar},
00037     // Group 2, Command 5
00038     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(6)), handle_get_current_charge_total},
00039     // Group 2, Command 6
00040     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(7)), handle_get_current_draw},
00041     // Group 2, Command 7
00042     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(0)), handle_time},
00043     // Group 3, Command 0
00044     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(1)), handle_timezone_offset},
00045     // Group 3, Command 1
00046     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(2)), handle_clock_sync_interval},
00047     // Group 3, Command 2
00048     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(3)), handle_get_last_sync_time},
00049     // Group 3, Command 3
00050     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(1)), handle_get_last_events},
00051     // Group 5, Command 1
00052     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(2)), handle_get_event_count},
00053     // Group 5, Command 2
00054     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(0)), handle_list_files},
00055     // Group 6, Command 0
00056     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(1)), handle_file_download},
00057     // Group 6, Command 1
00058     {((static_cast<uint32_t>(6) << 8) | static_cast<uint32_t>(4)), handle_mount},
00059     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(1)), handle_gps_power_status},
00060     // Group 7, Command 1
00061     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(2)), handle_enable_gps_uart_passthrough},
00062     // Group 7, Command 3
00063     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(3)), handle_get_rmc_data},
00064     // Group 7, Command 3
00065     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(4)), handle_get_gga_data},
00066     // Group 7, Command 4
00067
00068 };
00069
00070
00071 } // end of CommandSystem group

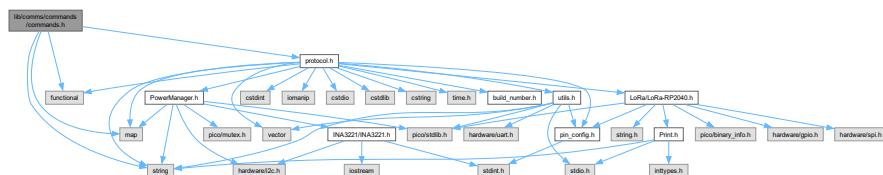
```

8.13 lib/comms/commands/commands.h File Reference

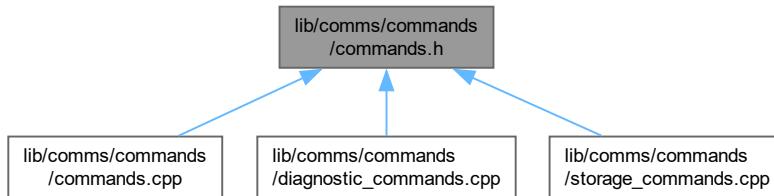
```

#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 ¶m, `OperationType` operationType)
Handler for getting and setting system time.
- `Frame handle_timezone_offset` (const std::string ¶m, `OperationType` operationType)
Handler for getting and setting timezone offset.
- `Frame handle_clock_sync_interval` (const std::string ¶m, `OperationType` operationType)
Handler for getting and setting clock synchronization interval.
- `Frame handle_get_last_sync_time` (const std::string ¶m, `OperationType` operationType)
Handler for getting last clock sync time.
- `Frame handle_get_commands_list` (const std::string ¶m, `OperationType` operationType)
Handler for listing all available commands on UART.
- `Frame handle_get_build_version` (const std::string ¶m, `OperationType` operationType)
Get firmware build version.
- `Frame handle_verbosity` (const std::string ¶m, `OperationType` operationType)
Handles setting or getting the UART verbosity level.
- `Frame handle_enter_bootloader_mode` (const std::string ¶m, `OperationType` operationType)
Reboot system to USB firmware loader.
- `Frame handle_gps_power_status` (const std::string ¶m, `OperationType` operationType)
Handler for controlling GPS module power state.
- `Frame handle_enable_gps_uart_passthrough` (const std::string ¶m, `OperationType` operationType)
Handler for enabling GPS transparent mode (UART pass-through)
- `Frame handle_get_rmc_data` (const std::string ¶m, `OperationType` operationType)
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- `Frame handle_get_gga_data` (const std::string ¶m, `OperationType` operationType)
Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.
- `Frame handle_get_power_manager_ids` (const std::string ¶m, `OperationType` operationType)
Handler for retrieving Power Manager IDs.
- `Frame handle_get_voltage_battery` (const std::string ¶m, `OperationType` operationType)
Handler for getting battery voltage.
- `Frame handle_get_voltage_5v` (const std::string ¶m, `OperationType` operationType)
Handler for getting 5V rail voltage.
- `Frame handle_get_current_charge_usb` (const std::string ¶m, `OperationType` operationType)
Handler for getting USB charge current.
- `Frame handle_get_current_charge_solar` (const std::string ¶m, `OperationType` operationType)
Handler for getting solar panel charge current.
- `Frame handle_get_current_charge_total` (const std::string ¶m, `OperationType` operationType)

- Frame handle_get_current_draw (const std::string ¶m, OperationType operationType)

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

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

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

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

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

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

Handles the SD card mount/unmount command.
- Frame execute_command (uint32_t commandKey, const std::string ¶m, OperationType operationType)

Executes a command based on its key.

Variables

- std::map< uint32_t, std::function< Frame(const std::string &, OperationType)> > commandHandlers

Global map of all command handlers.

8.14 commands.h

Go to the documentation of this file.

```

00001 // commands/commands.h
00002 #ifndef COMMANDS_H
00003 #define COMMANDS_H
00004
00005 #include <string>
00006 #include <functional>
00007 #include <map>
00008 #include "protocol.h"
00009
00010 // CLOCK
00011 Frame handle_time(const std::string& param, OperationType operationType);
00012 Frame handle_timezone_offset(const std::string& param, OperationType operationType);
00013 Frame handle_clock_sync_interval(const std::string& param, OperationType operationType);
00014 Frame handle_get_last_sync_time(const std::string& param, OperationType operationType);
00015
00016 // DIAG
00017 Frame handle_get_commands_list(const std::string& param, OperationType operationType);
00018 Frame handle_get_build_version(const std::string& param, OperationType operationType);
00019 Frame handle_verbosity(const std::string& param, OperationType operationType);
00020 Frame handle_enter_bootloader_mode(const std::string& param, OperationType operationType);
00021
00022 // GPS
00023 Frame handle_gps_power_status(const std::string& param, OperationType operationType);
00024 Frame handle_enable_gps_uart_passthrough(const std::string& param, OperationType operationType);
00025 Frame handle_get_rmc_data(const std::string& param, OperationType operationType);
00026 Frame handle_get_gga_data(const std::string& param, OperationType operationType);
00027
00028 // POWER
00029 Frame handle_get_power_manager_ids(const std::string& param, OperationType operationType);
00030 Frame handle_get_voltage_battery(const std::string& param, OperationType operationType);
00031 Frame handle_get_voltage_5v(const std::string& param, OperationType operationType);
00032 Frame handle_get_current_charge_usb(const std::string& param, OperationType operationType);
00033 Frame handle_get_current_charge_solar(const std::string& param, OperationType operationType);
00034 Frame handle_get_current_charge_total(const std::string& param, OperationType operationType);
00035 Frame handle_get_current_draw(const std::string& param, OperationType operationType);
00036
00037 // EVENT
00038 Frame handle_get_last_events(const std::string& param, OperationType operationType);
00039 Frame handle_get_event_count(const std::string& param, OperationType operationType);
00040
00041
00042 //STORAGE

```

```

00043 Frame handle_list_files(const std::string& param, OperationType operationType);
00044 Frame handle_file_download(const std::string& param, OperationType operationType);
00045 Frame handle_mount(const std::string& param, OperationType operationType);
00046
00047 Frame execute_command(uint32_t commandKey, const std::string& param, OperationType operationType);
00048 extern std::map<uint32_t, std::function<Frame(const std::string&, OperationType)>> commandHandlers;
00049
00050 #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 ¶m, OperationType operationType)**
Handler for listing all available commands on UART.
- **Frame handle_get_build_version (const std::string ¶m, OperationType operationType)**
Get firmware build version.
- **Frame handle_verbosity (const std::string ¶m, OperationType operationType)**
Handles setting or getting the UART verbosity level.
- **Frame handle_enter_bootloader_mode (const std::string ¶m, OperationType operationType)**
Reboot system to USB firmware loader.

8.16 diagnostic_commands.cpp

Go to the documentation of this file.

```

00001 #include "communication.h"
00002 #include "commands.h"
00003 #include "pico/stdlib.h"
00004 #include "pico/bootrom.h"
00005
00010
00021 Frame handle_get_commands_list(const std::string& param, OperationType operationType) {
00022     if (!param.empty()) {
00023         return frame_build(ExecutionResult::ERROR, 1, 0, "PARAM UNNECESSARY");
00024     }
00025
00026     if (!(operationType == OperationType::GET)) {
00027         return frame_build(ExecutionResult::ERROR, 1, 0, "INVALID OPERATION");
00028     }
00029
00030     std::stringstream ss;
00031     for (const auto& entry : commandHandlers) {
00032         uint32_t commandKey = entry.first;
00033         uint8_t group = (commandKey >> 8) & 0xFF;
00034         uint8_t command = commandKey & 0xFF;
00035
00036         ss << "Group: " << static_cast<int>(group)

```

```

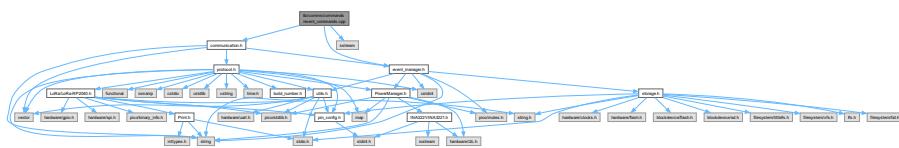
00037         << ", Command: " << static_cast<int>(command) << "\n";
00038     }
00039
00040     std::string commandList = ss.str();
00041     uart_print(commandList, VerbosityLevel::INFO); // Print to UART
00042
00043     return frame_build(ExecutionResult::SUCCESS, 1, 0, "Commands listed on UART");
00044 }
00045
00056 Frame handle_get_build_version(const std::string& param, OperationType operationType) {
00057     if (!param.empty()) {
00058         return frame_build(ExecutionResult::ERROR, 1, 1, "PARAM UNECESSARY");
00059     }
00060     if (operationType == OperationType::GET) {
00061         return frame_build(ExecutionResult::SUCCESS, 1, 1, std::to_string(BUILD_NUMBER));
00062     }
00063     return frame_build(ExecutionResult::ERROR, 1, 1, "INVALID OPERATION");
00064 }
00065
00066
00088 Frame handle_verbosity(const std::string& param, OperationType operationType) {
00089     if (param.empty()) {
00090         uart_print("Current verbosity level: " + std::to_string(static_cast<int>(g_uart_verbosity)),
00091             VerbosityLevel::INFO);
00092         return frame_build(ExecutionResult::SUCCESS, 1, 8,
00093                             std::to_string(static_cast<int>(g_uart_verbosity)));
00094     }
00095     try {
00096         int level = std::stoi(param);
00097         if (level < 0 || level > 5) {
00098             return frame_build(ExecutionResult::ERROR, 1, 8, "INVALID LEVEL (0-5)");
00099         }
00100         g_uart_verbosity = static_cast<VerbosityLevel>(level);
00101         return frame_build(ExecutionResult::SUCCESS, 1, 8, "LEVEL SET");
00102     } catch (...) {
00103         return frame_build(ExecutionResult::ERROR, 1, 8, "INVALID FORMAT");
00104     }
00105 }
00106
00117 Frame handle_enter_bootloader_mode(const std::string& param, OperationType operationType) {
00118     if (!param.empty()) {
00119         return frame_build(ExecutionResult::ERROR, 1, 9, "PARAM UNNECESSARY");
00120     }
00121
00122     if (operationType != OperationType::SET) {
00123         return frame_build(ExecutionResult::ERROR, 1, 9, "INVALID OPERATION");
00124     }
00125
00126     // Build the success frame *before* resetting
00127     Frame successFrame = frame_build(ExecutionResult::SUCCESS, 1, 9, "REBOOT BOOTSEL");
00128
00129     // Send the success frame
00130     uart_print("Sending BOOTSEL confirmation...");
00131     send_frame(successFrame); // Assuming you have a sendFrame function
00132
00133     // Delay to ensure the frame is sent
00134     sleep_ms(100);
00135
00136     uart_print("Entering BOOTSEL mode...", VerbosityLevel::WARNING);
00137     reset_usb_boot(0, 0); // Trigger BOOTSEL mode
00138
00139     // The code will never reach here because the Pico will reset
00140     return frame_build(ExecutionResult::SUCCESS, 1, 9, "Entering BOOTSEL mode");
00141 }
00142

```

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

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

Include dependency graph for event_commands.cpp:



Functions

- **Frame handle_get_last_events (const std::string ¶m, OperationType operationType)**
Handler for retrieving last N events from the event log.
- **Frame handle_get_event_count (const std::string ¶m, OperationType operationType)**
Handler for getting total number of events in the log.

8.18 event_commands.cpp

Go to the documentation of this file.

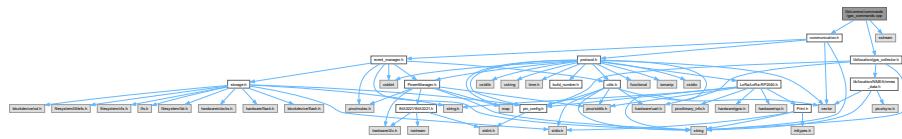
```

00001 #include "communication.h"
00002 #include "event_manager.h"
00003 #include <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: IIIITTTTTTGEE
00056         // IIII: 16-bit ID (4 hex chars)
00057         // TTTTTT: 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 ¶m, OperationType operationType)**
Handler for controlling GPS module power state.
- **Frame handle_enable_gps_uart_passthrough (const std::string ¶m, OperationType operationType)**
Handler for enabling GPS transparent mode (UART pass-through)
- **Frame handle_get_rmc_data (const std::string ¶m, OperationType operationType)**
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- **Frame handle_get_gga_data (const std::string ¶m, OperationType operationType)**
Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.

8.20 gps_commands.cpp

Go to the documentation of this file.

```
00001 #include "communication.h"
00002 #include "lib/location/gps_collector.h"
00003 #include <sstream> // Include for stringstream
00004
00010
00026 Frame handle_gps_power_status(const std::string& param, OperationType operationType) {
00027     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00028         return frame_build(ExecutionResult::ERROR, 7, 1, "INVALID OPERATION");
00029     }
00030
00031     if (operationType == OperationType::SET) {
00032         if (param.empty()) {
00033             return frame_build(ExecutionResult::ERROR, 7, 1, "PARAM REQUIRED");
00034         }
00035
00036         try {
00037             int powerStatus = std::stoi(param);
00038             if (powerStatus != 0 && powerStatus != 1) {
00039                 return frame_build(ExecutionResult::ERROR, 7, 1, "INVALID VALUE. USE 0 OR 1");
00040             }
00041             gpio_put(GPS_POWER_ENABLE_PIN, powerStatus);
00042             EventEmitter::emit(EventGroup::GPS, powerStatus ? GPSEvent::POWER_ON :
00043                             GPSEvent::POWER_OFF);
00044             return frame_build(ExecutionResult::SUCCESS, 7, 1, std::to_string(powerStatus));
00045         } catch (...) {
00046             return frame_build(ExecutionResult::ERROR, 7, 1, "INVALID PARAMETER FORMAT");
00047         }
00048
00049     // GET operation
00050     if (!param.empty()) {
00051         return frame_build(ExecutionResult::ERROR, 7, 1, "PARAM UNNECESSARY");
00052     }
00053
00054     bool powerStatus = gpio_get(GPS_POWER_ENABLE_PIN);
00055     return frame_build(ExecutionResult::SUCCESS, 7, 1, std::to_string(powerStatus));
```

```

00056 }
00057
00058
00074 Frame handle_enable_gps_uart_passthrough(const std::string& param, OperationType operationType) {
00075     // Validate operation type
00076     if (!operationType == OperationType::SET) {
00077         return frame_build(ExecutionResult::ERROR, 7, 2, "NOT ALLOWED");
00078     }
00079
00080     // Parse and validate timeout parameter
00081     uint32_t timeoutMs;
00082     try {
00083         timeoutMs = param.empty() ? 60000u : std::stoul(param) * 1000;
00084     } catch (...) {
00085         return frame_build(ExecutionResult::ERROR, 7, 2, "INVALID TIMEOUT FORMAT");
00086     }
00087
00088     // Setup UART parameters and exit sequence
00089     const std::string EXIT_SEQUENCE = "##EXIT##";
00090     std::string inputBuffer;
00091     bool exitRequested = false;
00092     uint32_t originalBaudRate = DEBUG_UART_BAUD_RATE;
00093     uint32_t gpsBaudRate = GPS_UART_BAUD_RATE;
00094     uint32_t startTime = to_ms_since_boot(get_absolute_time());
00095
00096     // Log start of transparent mode
00097     EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_START);
00098
00099     // Print startup message
00100     std::string message = "Entering GPS Serial Pass-Through Mode @" +
00101             std::to_string(gpsBaudRate) + " for " +
00102             std::to_string(timeoutMs/1000) + "s\r\n" +
00103             "Send " + EXIT_SEQUENCE + " to exit";
00104     uart_print(message, VerbosityLevel::INFO);
00105
00106     // Allow time for message to be sent before baudrate change
00107     sleep_ms(10);
00108
00109     // Switch to GPS baudrate
00110     uart_set_baudrate(DEBUG_UART_PORT, gpsBaudRate);
00111
00112     // Main transparent mode loop
00113     while (!exitRequested) {
00114         while (uart_is_readable(DEBUG_UART_PORT)) {
00115             char ch = uart_getc(DEBUG_UART_PORT);
00116
00117             inputBuffer += ch;
00118             if (inputBuffer.length() > EXIT_SEQUENCE.length()) {
00119                 inputBuffer = inputBuffer.substr(1);
00120             }
00121
00122             if (inputBuffer == EXIT_SEQUENCE) {
00123                 exitRequested = true;
00124                 break;
00125             }
00126
00127             if (inputBuffer != EXIT_SEQUENCE.substr(0, inputBuffer.length())) {
00128                 uart_write_blocking(GPS_UART_PORT,
00129                     reinterpret_cast<const uint8_t*>(&ch), 1);
00130             }
00131         }
00132
00133         while (uart_is_readable(GPS_UART_PORT)) {
00134             char gpsByte = uart_getc(GPS_UART_PORT);
00135             uart_write_blocking(DEBUG_UART_PORT,
00136                 reinterpret_cast<const uint8_t*>(&gpsByte), 1);
00137         }
00138
00139         if (to_ms_since_boot(get_absolute_time()) - startTime >= timeoutMs) {
00140             break;
00141         }
00142     }
00143
00144     uart_set_baudrate(DEBUG_UART_PORT, originalBaudRate);
00145
00146     sleep_ms(10);
00147
00148     EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_END);
00149
00150     std::string exitReason = exitRequested ? "USER_EXIT" : "TIMEOUT";
00151     std::string response = "GPS UART BRIDGE EXIT: " + exitReason;
00152     uart_print(response, VerbosityLevel::INFO);
00153
00154     return frame_build(ExecutionResult::SUCCESS, 7, 2, response);
00155 }
00156
00157

```

```
00170 Frame handle_get_rmc_data(const std::string& param, OperationType operationType) {
00171     if (operationType != OperationType::GET) {
00172         return frame_build(ExecutionResult::ERROR, 7, 3, "INVALID OPERATION");
00173     }
00174
00175     if (!param.empty()) {
00176         return frame_build(ExecutionResult::ERROR, 7, 3, "PARAM UNNECESSARY");
00177     }
00178
00179     std::vector<std::string> tokens = nmea_data.get_rmc_tokens();
00180     if (tokens.empty()) {
00181         return frame_build(ExecutionResult::ERROR, 7, 3, "NO RMC DATA");
00182     }
00183
00184     // Join tokens with commas to create the response
00185     std::stringstream ss;
00186     for (size_t i = 0; i < tokens.size(); ++i) {
00187         ss << tokens[i];
00188         if (i < tokens.size() - 1) {
00189             ss << ",";
00190         }
00191     }
00192
00193     return frame_build(ExecutionResult::SUCCESS, 7, 3, ss.str());
00194 }
00195
00196
00197 00209 Frame handle_get_gga_data(const std::string& param, OperationType operationType) {
00210     if (operationType != OperationType::GET) {
00211         return frame_build(ExecutionResult::ERROR, 7, 4, "INVALID OPERATION");
00212     }
00213
00214     if (!param.empty()) {
00215         return frame_build(ExecutionResult::ERROR, 7, 4, "PARAM UNNECESSARY");
00216     }
00217
00218     std::vector<std::string> tokens = nmea_data.get_gga_tokens();
00219     if (tokens.empty()) {
00220         return frame_build(ExecutionResult::ERROR, 7, 4, "NO GGA DATA");
00221     }
00222
00223     // Join tokens with commas to create the response
00224     std::stringstream ss;
00225     for (size_t i = 0; i < tokens.size(); ++i) {
00226         ss << tokens[i];
00227         if (i < tokens.size() - 1) {
00228             ss << ",";
00229         }
00230     }
00231
00232     return frame_build(ExecutionResult::SUCCESS, 7, 4, ss.str());
00233 } // 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 \(const std::string ¶m, OperationType operationType\)](#)
Handler for retrieving Power Manager IDs.
- [Frame handle_get_voltage_battery \(const std::string ¶m, OperationType operationType\)](#)
Handler for getting battery voltage.

- Frame `handle_get_voltage_5v` (const std::string ¶m, OperationType operationType)
Handler for getting 5V rail voltage.
- Frame `handle_get_current_charge_usb` (const std::string ¶m, OperationType operationType)
Handler for getting USB charge current.
- Frame `handle_get_current_charge_solar` (const std::string ¶m, OperationType operationType)
Handler for getting solar panel charge current.
- Frame `handle_get_current_charge_total` (const std::string ¶m, OperationType operationType)
Handler for getting total charge current.
- Frame `handle_get_current_draw` (const std::string ¶m, OperationType operationType)
Handler for getting system current draw.

8.22 power_commands.cpp

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

```

00001 #include "communication.h"
00002
00003
00004
00005
00006
00007
00008
00009
00010 Frame handle_get_power_manager_ids(const std::string& param, OperationType operationType) {
00011     if (!param.empty()) {
00012         return frame_build(ExecutionResult::ERROR, 2, 0, "PARAM UNNECESSARY");
00013     }
00014
00015     if (!(operationType == OperationType::GET)) {
00016         return frame_build(ExecutionResult::ERROR, 2, 0, "INVALID OPERATION");
00017     }
00018
00019     extern PowerManager powerManager;
00020     std::string powerManagerIDS = powerManager.read_device_ids();
00021     return frame_build(ExecutionResult::SUCCESS, 2, 0, powerManagerIDS);
00022 }
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032
00033
00034
00035
00036
00037
00038
00039 Frame handle_get_voltage_battery(const std::string& param, OperationType operationType) {
00040     if (!param.empty()) {
00041         return frame_build(ExecutionResult::ERROR, 2, 2, "PARAM UNNECESSARY");
00042     }
00043
00044     if (!(operationType == OperationType::GET)) {
00045         return frame_build(ExecutionResult::ERROR, 2, 2, "NOT ALLOWED");
00046     }
00047
00048     extern PowerManager powerManager;
00049     float voltage = powerManager.get_voltage_battery();
00050     return frame_build(ExecutionResult::SUCCESS, 2, 2, std::to_string(voltage), ValueUnit::VOLT);
00051 }
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067 Frame handle_get_voltage_5v(const std::string& param, OperationType operationType) {
00068     if (!param.empty()) {
00069         return frame_build(ExecutionResult::ERROR, 2, 3, "PARAM UNNECESSARY");
00070     }
00071
00072     if (!(operationType == OperationType::GET)) {
00073         return frame_build(ExecutionResult::ERROR, 2, 3, "NOT ALLOWED");
00074     }
00075
00076     extern PowerManager powerManager;
00077     float voltage = powerManager.get_voltage_5v();
00078     return frame_build(ExecutionResult::SUCCESS, 2, 3, std::to_string(voltage), ValueUnit::VOLT);
00079 }
00080
00081
00082
00083
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093 Frame handle_get_current_charge_usb(const std::string& param, OperationType operationType) {
00094     if (!param.empty()) {
00095         return frame_build(ExecutionResult::ERROR, 2, 4, "PARAM UNNECESSARY");
00096     }
00097
00098     if (!(operationType == OperationType::GET)) {
00099         return frame_build(ExecutionResult::ERROR, 2, 4, "NOT ALLOWED");
00100     }
00101
00102     extern PowerManager powerManager;
00103     float chargeCurrent = powerManager.get_current_charge_usb();
00104 }
```

```

00114     return frame_build(ExecutionResult::SUCCESS, 2, 4, std::to_string(chargeCurrent),
00115     ValueUnit::MILIAMP);
00116
00117
00118 Frame handle_get_current_charge_solar(const std::string& param, OperationType operationType) {
00119     if (!param.empty()) {
00120         return frame_build(ExecutionResult::ERROR, 2, 5, "PARAM UNNECESSARY");
00121     }
00122
00123     if (!(operationType == OperationType::GET)) {
00124         return frame_build(ExecutionResult::ERROR, 2, 5, "NOT ALLOWED");
00125     }
00126
00127     extern PowerManager powerManager;
00128     float chargeCurrent = powerManager.get_current_charge_solar();
00129     return frame_build(ExecutionResult::SUCCESS, 2, 5, std::to_string(chargeCurrent),
00130     ValueUnit::MILIAMP);
00131 }
00132
00133
00134
00135 Frame handle_get_current_charge_total(const std::string& param, OperationType operationType) {
00136     if (!param.empty()) {
00137         return frame_build(ExecutionResult::ERROR, 2, 6, "PARAM UNNECESSARY");
00138     }
00139
00140     if (!(operationType == OperationType::GET)) {
00141         return frame_build(ExecutionResult::ERROR, 2, 6, "NOT ALLOWED");
00142     }
00143
00144
00145 Frame handle_get_current_draw(const std::string& param, OperationType operationType) {
00146     if (!param.empty()) {
00147         return frame_build(ExecutionResult::ERROR, 2, 7, "PARAM UNNECESSARY");
00148     }
00149
00150
00151
00152     if (!(operationType == OperationType::GET)) {
00153         return frame_build(ExecutionResult::ERROR, 2, 7, "NOT ALLOWED");
00154     }
00155
00156     extern PowerManager powerManager;
00157     float currentDraw = powerManager.get_current_draw();
00158     return frame_build(ExecutionResult::SUCCESS, 2, 7, std::to_string(currentDraw),
00159     ValueUnit::MILIAMP);
00160 }
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
00181
00182
00183
00184
00185
00186
00187
00188
00189
00190
00191
00192
00193
00194
00195
00196
00197 } // 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 ¶m, OperationType operationType)
Handles the file download command.
- Frame handle_list_files (const std::string ¶m, OperationType operationType)
Handles the list files command.
- Frame handle_mount (const std::string ¶m, OperationType operationType)
Handles the SD card mount/unmount command.

8.23.1 Macro Definition Documentation

8.23.1.1 MAX_BLOCK_SIZE

```
#define MAX_BLOCK_SIZE 250
```

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

8.23.1.2 STORAGE_GROUP

```
#define STORAGE_GROUP 6
```

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

8.23.1.3 START_COMMAND

```
#define START_COMMAND 1
```

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

8.23.1.4 DATA_COMMAND

```
#define DATA_COMMAND 2
```

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

8.23.1.5 END_COMMAND

```
#define END_COMMAND 3
```

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

8.23.1.6 LIST_FILES_COMMAND

```
#define LIST_FILES_COMMAND 0
```

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

8.23.1.7 MOUNT_COMMAND

```
#define MOUNT_COMMAND 4
```

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

8.24 storage_commands.cpp

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

```
00001 #include "commands.h"
00002 #include "communication.h"
00003 #include "storage.h"
00004 #include "filesystem/vfs.h"
00005 #include "filesystem/littlefs.h"
00006 #include <sys/stat.h>
00007 #include <errno.h>
00008 #include "storage_commands_utils.h"
00009 #include "dirent.h"
0010
0011 #define MAX_BLOCK_SIZE 250
0012 #define STORAGE_GROUP 6
0013 #define START_COMMAND 1
0014 #define DATA_COMMAND 2
0015 #define END_COMMAND 3
0016 #define LIST_FILES_COMMAND 0
0017 #define MOUNT_COMMAND 4
0023
0043 Frame handle_file_download(const std::string& param, OperationType operationType) {
0044     if (operationType != OperationType::GET) {
0045         return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, START_COMMAND, "Invalid operation
type");
0046     }
0047
0048     const char* filename = param.c_str();
0049     FILE* file = fopen(filename, "rb");
0050
0051     if (!file) {
0052         return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, START_COMMAND, "File not found");
0053     }
0054
0055     // Get file size
0056     fseek(file, 0, SEEK_END);
0057     size_t fileSize = ftell(file);
0058     fseek(file, 0, SEEK_SET);
0059
0060     // Send file size to ground station
0061     Frame sizeFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, START_COMMAND,
std::to_string(fileSize));
0062     send_frame(sizeFrame);
0063
0064     size_t block_size = MAX_BLOCK_SIZE;
0065     size_t block_count = (fileSize + block_size - 1) / block_size;
0066
0067     // Send block size and count
```

```

00068     Frame blockSizeFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, START_COMMAND,
00069         std::to_string(block_size));
00070     send_frame(blockSizeFrame);
00071     Frame blockCountFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, START_COMMAND,
00072         std::to_string(block_count));
00073     send_frame(blockCountFrame);
00074     uint8_t buffer[MAX_BLOCK_SIZE];
00075     size_t bytesRead;
00076     uint32_t totalChecksum = 0;
00077     size_t blockIndex = 0;
00078     while ((bytesRead = fread(buffer, 1, MAX_BLOCK_SIZE, file)) > 0) {
00079         // Send data block
00080         send_data_block(buffer, bytesRead);
00081         totalChecksum = calculate_checksum(buffer, bytesRead);
00082         // Wait for ACK
00083         if (!receive_ack()) {
00084             fclose(file);
00085             return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, DATA_COMMAND, "ACK timeout");
00086         }
00087         blockIndex++;
00088     }
00089 }
00090 fclose(file);
00091 // Send end frame with checksum
00092 std::stringstream ss;
00093 ss << std::hex << totalChecksum;
00094 Frame endFrame = frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, END_COMMAND, ss.str());
00095 send_frame(endFrame);
00096 return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, END_COMMAND, "File download
complete");
00100 }
00101
00102
00120 Frame handle_list_files(const std::string& param, OperationType operationType) {
00121     if (operationType != OperationType::GET) {
00122         return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, LIST_FILES_COMMAND, "Invalid
operation type");
00123     }
00124
00125     DIR *dir;
00126     struct dirent *ent;
00127     if ((dir = opendir("/")) != NULL) {
00128         while ((ent = readdir(dir)) != NULL) {
00129             const char* filename = ent->d_name;
00130
00131             // Skip "." and ".." directories
00132             if (strcmp(filename, ".") == 0 || strcmp(filename, "..") == 0) {
00133                 continue;
00134             }
00135
00136             // Get file size
00137             char filepath[256];
00138             snprintf(filepath, sizeof(filepath), "/%s", filename);
00139
00140             FILE* file = fopen(filepath, "rb");
00141             fileSize = 0;
00142
00143             if (file != NULL) {
00144                 fseek(file, 0, SEEK_END);
00145                 fileSize = ftell(file);
00146                 fclose(file);
00147             }
00148
00149             // Create and send frame with filename and size
00150             char fileInfo[512];
00151             snprintf(fileInfo, sizeof(fileInfo), "%s:%zu", filename, fileSize);
00152             Frame fileFrame = frame_build(ExecutionResult::INFO, STORAGE_GROUP, LIST_FILES_COMMAND,
00153                 fileInfo);
00154             uart_print(fileInfo, VerboseLevel::INFO);
00155             send_frame(fileFrame);
00156         }
00157         closedir(dir);
00158     }
00159     return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, LIST_FILES_COMMAND, "File listing
complete");
00160 } else {
00159     return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, LIST_FILES_COMMAND, "Could not open
directory");
00160 }
00161
00162
00163
00180 Frame handle_mount(const std::string& param, OperationType operationType) {

```

```

00181     if (operationType != OperationType::SET) {
00182         return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Invalid operation
00183         type");
00184
00185     if (param == "1") {
00186         if (fs_init()) {
00187             return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, MOUNT_COMMAND, "SD card
00188             mounted");
00189         } else {
00190             return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Mount failed");
00191         }
00192     } else if (param == "0") {
00193         if (fs_unmount("/")) {
00194             sd_card_mounted = false;
00195             return frame_build(ExecutionResult::SUCCESS, STORAGE_GROUP, MOUNT_COMMAND, "SD card
00196             unmounted");
00197         } else {
00198             return frame_build(ExecutionResult::ERROR, STORAGE_GROUP, MOUNT_COMMAND, "Unmount
00199             failed");
00200         }
00201     }
00202 } // StorageCommands

```

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

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



Functions

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

8.25.1 Function Documentation

8.25.1.1 calculate_checksum()

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

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

Here is the caller graph for this function:



8.25.1.2 send_data_block()

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

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

Here is the caller graph for this function:



8.25.1.3 receive_ack()

```
bool receive_ack ()
```

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

Here is the caller graph for this function:



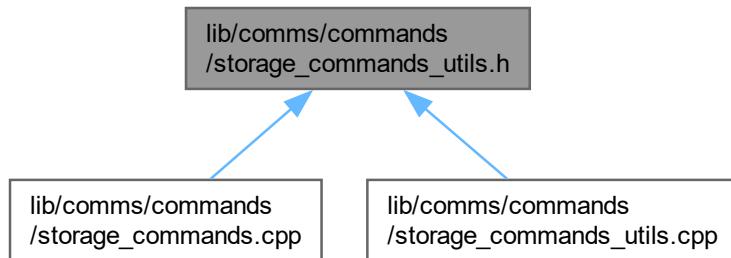
8.26 storage_commands_utils.cpp

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

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

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

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



Functions

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

8.27.1 Function Documentation

8.27.1.1 calculate_checksum()

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

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

Here is the caller graph for this function:



8.27.1.2 send_data_block()

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

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

Here is the caller graph for this function:



8.27.1.3 receive_ack()

```
bool receive_ack ()
```

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

Here is the caller graph for this function:



8.28 storage_commands_utils.h

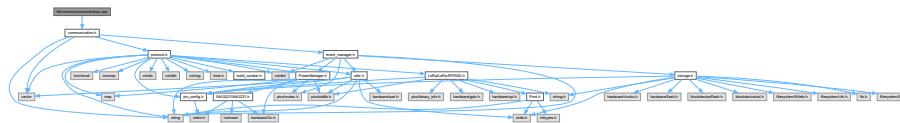
Go to the documentation of this file.

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

8.29 lib/comms/communication.cpp File Reference

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

Include dependency graph for communication.cpp:



Functions

- bool `initialize_radio()`
Initializes the LoRa radio module.

Variables

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

8.29.1 Function Documentation

8.29.1.1 initialize_radio()

```
bool initialize_radio ()
```

Initializes the LoRa radio module.

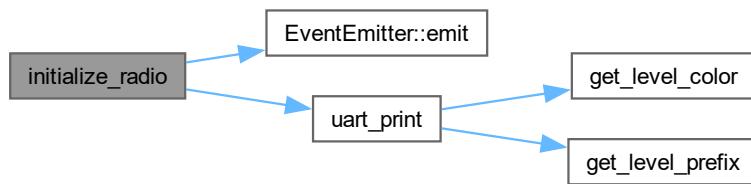
Returns

True if initialization was successful, false otherwise.

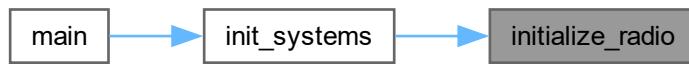
Sets the LoRa pins and attempts to begin LoRa communication at a specified frequency. Emits a [CommsEvent::RADIO_INIT](#) event on success or a [CommsEvent::RADIO_ERROR](#) event on failure.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.29.2 Variable Documentation

8.29.2.1 outgoing

```
string outgoing
```

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

8.29.2.2 msgCount

```
uint8_t msgCount = 0
```

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

8.29.2.3 lastSendTime

```
long lastSendTime = 0
```

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

8.29.2.4 lastReceiveTime

```
long lastReceiveTime = 0
```

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

8.29.2.5 lastPrintTime

```
long lastPrintTime = 0
```

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

8.29.2.6 interval

```
unsigned long interval = 0
```

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

8.30 communication.cpp

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

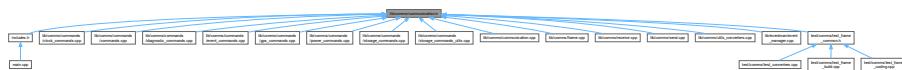
```
00001 #include "communication.h"
00002
00003 string outgoing;
00004 uint8_t msgCount = 0;
00005 long lastSendTime = 0;
00006 long lastReceiveTime = 0;
00007 long lastPrintTime = 0;
00008 unsigned long interval = 0;
00009
00010
00011 bool initialize_radio() {
00012     LoRa.set_pins(lora_cs_pin, lora_reset_pin, lora_irq_pin);
00013     long frequency = 433E6;
00014     bool initStatus = false;
00015     if (!LoRa.begin(frequency))
00016     {
00017         uart_print("LoRa init failed. Check your connections.", VerbosityLevel::WARNING);
00018         initStatus = false;
00019     } else {
00020         uart_print("LoRa initialized with frequency " + std::to_string(frequency),
00021             VerbosityLevel::INFO);
00022         initStatus = true;
00023     }
00024
00025     EventEmitter::emit(EventGroup::COMMS, initStatus ? CommsEvent::RADIO_INIT :
00026         CommsEvent::RADIO_ERROR);
00027
00028 }
00029
00030
00031     return initStatus;
00032 }
00033 }
```

8.31 lib/comms/communication.h File Reference

```
#include <string>
#include <vector>
#include "protocol.h"
#include "event_manager.h"
Include dependency graph for communication.h:
```



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



Functions

- `bool initialize_radio ()`
Initializes the LoRa radio module.
- `void on_receive (int packetSize)`
Callback function for handling received LoRa packets.
- `void handle_uart_input ()`
Handles UART input.
- `void send_message (std::string outgoing)`
- `void send_frame (const Frame &frame)`
- `void send_frame_uart (const Frame &frame)`
- `void send_frame_lora (const Frame &frame)`
- `void split_and_send_message (const uint8_t *data, size_t length)`
Sends a large packet using LoRa.
- `Frame execute_command (uint32_t commandKey, const std::string ¶m, OperationType operationType)`
Executes a command based on its key.
- `void frame_process (const std::string &data, Interface interface)`
Executes a command based on the command key and the parameter.
- `std::string frame_encode (const Frame &frame)`
Encodes a `Frame` instance into a string.
- `Frame frame_decode (const std::string &data)`
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.

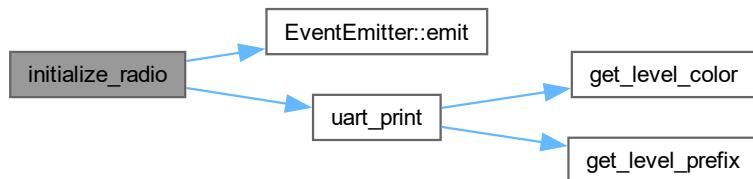
Returns

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.

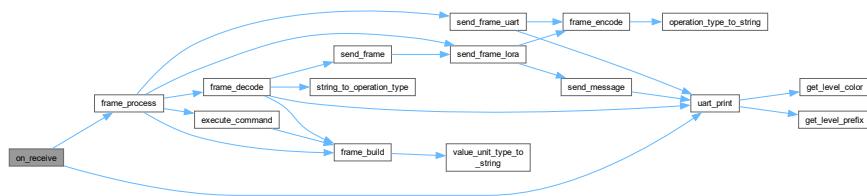
Parameters

<code>packetSize</code>	The size of the received packet.
-------------------------	----------------------------------

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.31.1.3 handle_uart_input()

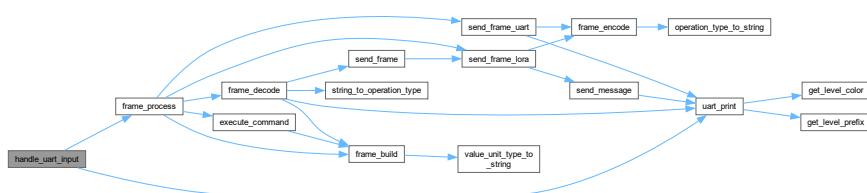
```
void handle_uart_input ()
```

Handles UART input.

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

Definition at line 76 of file [receive.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.31.1.4 send_message()

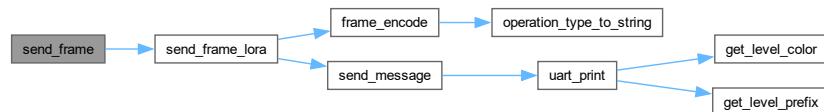
```
void send_message (
    std::string outgoing)
```

8.31.1.5 send_frame()

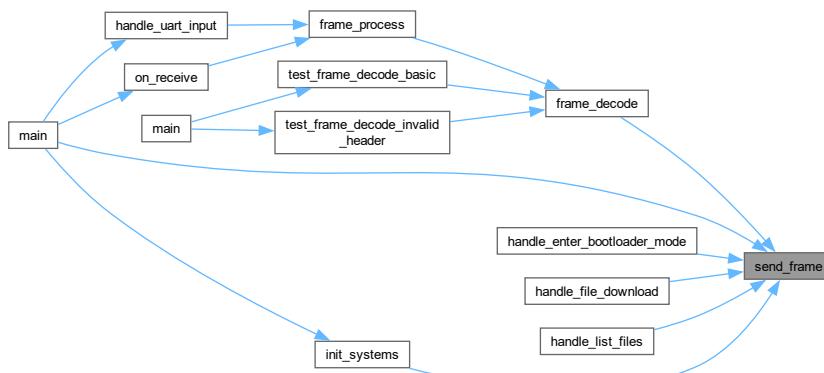
```
void send_frame (
    const Frame & frame)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

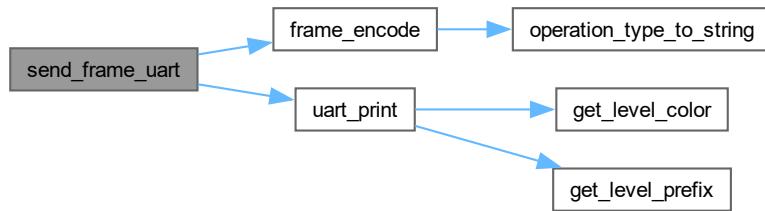


8.31.1.6 send_frame_uart()

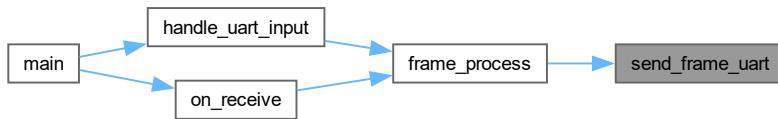
```
void send_frame_uart (
    const Frame & frame)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

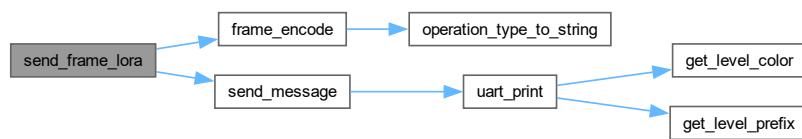


8.31.1.7 send_frame_lora()

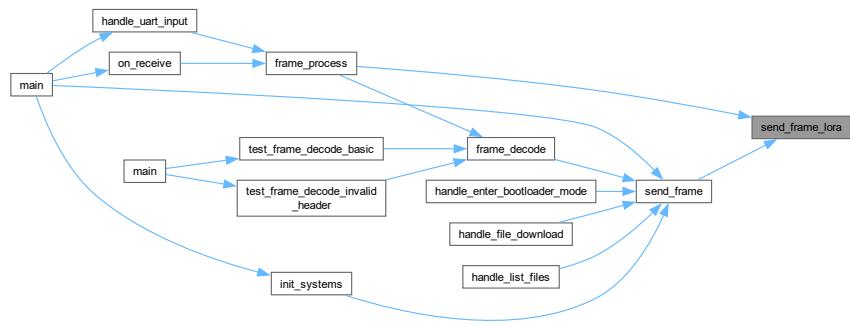
```
void send_frame_lora (
    const Frame & frame)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.31.1.8 split_and_send_message()

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

Sends a large packet using LoRa.

Parameters

<code>data</code>	The data to send.
<code>length</code>	The length of the data.

Splits the data into chunks of MAX_PKT_SIZE and sends each chunk as a separate LoRa packet.

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

8.31.1.9 frame_process()

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

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

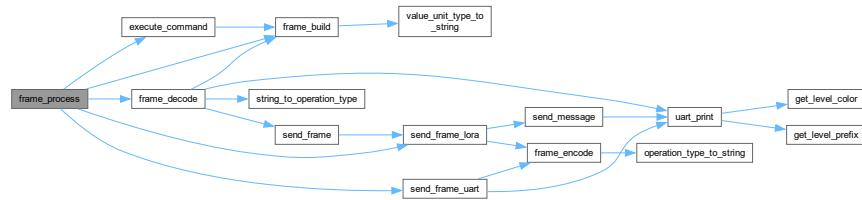
Parameters

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

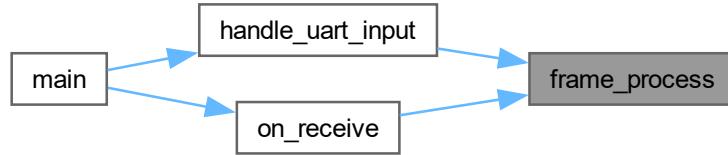
Decodes the frame data, extracts the command key, and executes the corresponding command. Sends the response frame. If an error occurs, an error frame is built and sent.

Definition at line 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

<i>frame</i>	The Frame instance to encode.
--------------	---

Returns

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

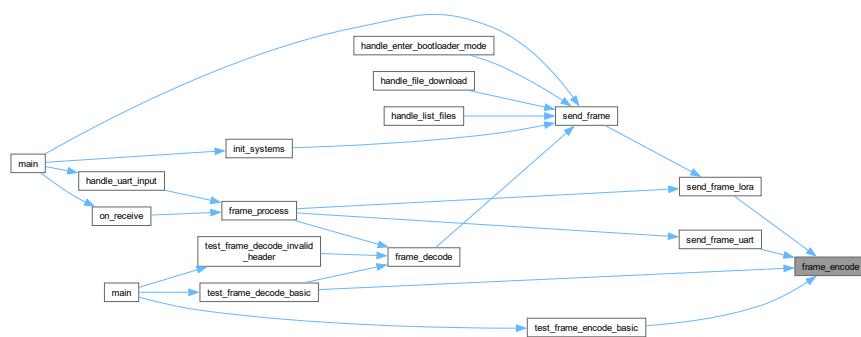
The encoded string includes the frame direction, operation type, group, command, value, and unit, all delimited by the DELIMITER character. The string is encapsulated by FRAME_BEGIN and FRAME_END.

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

<code>data</code>	The Frame data as a string.
-------------------	---

Returns

The [Frame](#) instance.

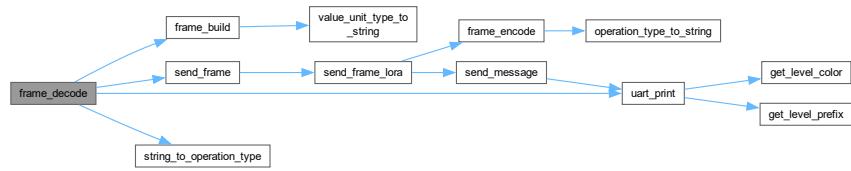
Exceptions

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

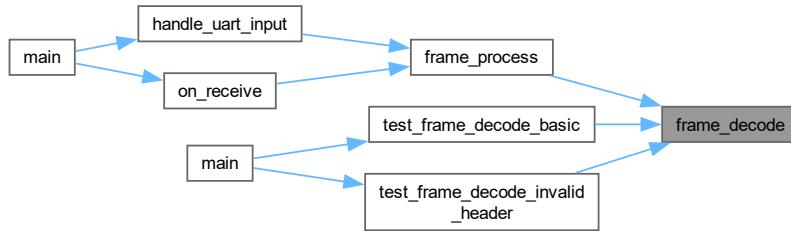
Parses the input string, extracting the frame direction, operation type, group, command, value, and unit. If an error occurs during parsing, an error message is printed to the UART, an error frame is built and sent, and the exception is re-thrown.

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

<i>result</i>	The execution result.
<i>group</i>	The group ID.
<i>command</i>	The command ID.
<i>value</i>	The value.
<i>unitType</i>	The value unit type.

Returns

The [Frame](#) instance.

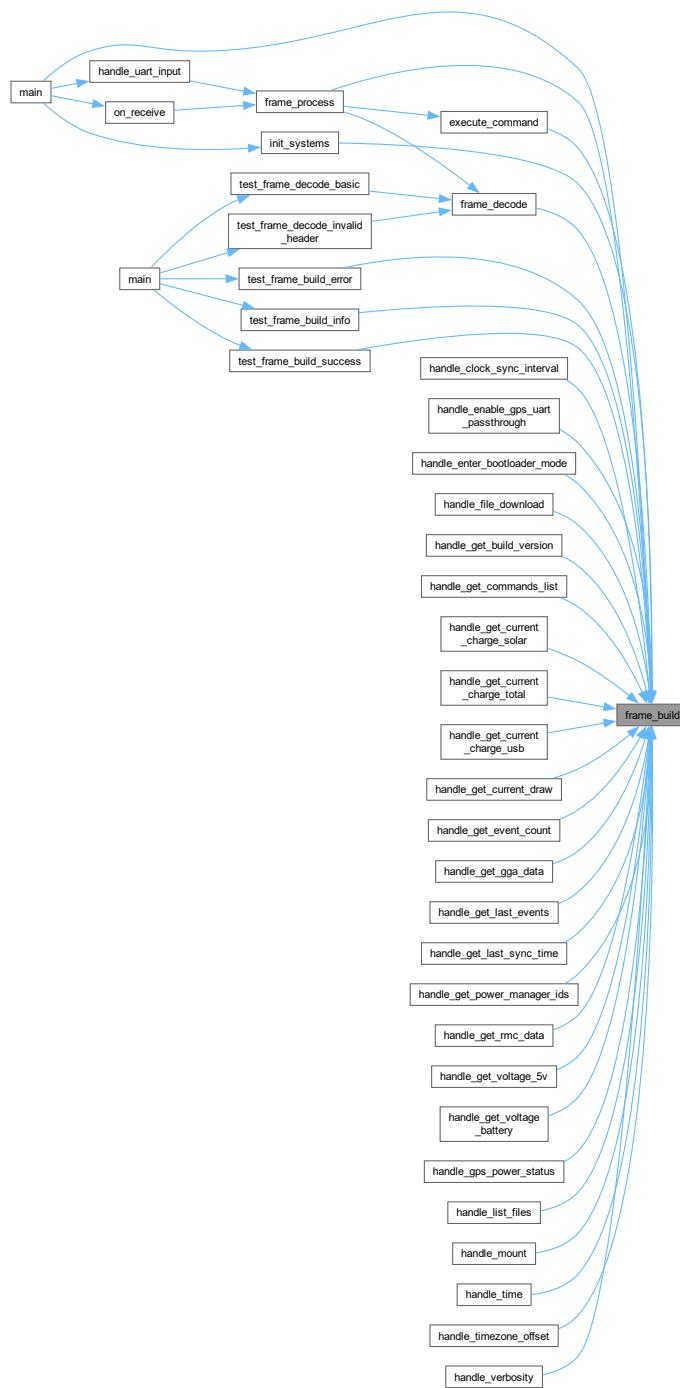
Constructs a [Frame](#) instance based on the provided parameters. The frame direction and operation type are set based on the execution result.

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.](#)

```

00001 #ifndef COMMUNICATION_H
00002 #define COMMUNICATION_H
00003
00004 #include <string>
00005 #include <vector>
00006 #include "protocol.h"
00007 #include "event_manager.h"
00008
00009 bool initialize_radio();
00010 void on_receive(int packetSize);
00011 void handle_uart_input();
00012 void send_message(std::string outgoing);
00013 void send_frame(const Frame& frame);
00014 void send_frame_uart(const Frame& frame);
00015 void send_frame_lora(const Frame& frame);
00016
00017 void split_and_send_message(const uint8_t* data, size_t length);
00018
00019 Frame execute_command(uint32_t commandKey, const std::string& param, OperationType operationType);
00020
00021 void frame_process(const std::string& data, Interface interface);
00022 std::string frame_encode(const Frame& frame);
00023 Frame frame_decode(const std::string& data);
00024 Frame frame_build(ExecutionResult result, uint8_t group, uint8_t command, const std::string& value,
00025   const ValueUnit unitType = ValueUnit::UNDEFINED);
00026
00027 std::string determine_unit(uint8_t group, uint8_t command);
00028 #endif

```

8.33 lib/comms/frame.cpp File Reference

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

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

Include dependency graph for frame.cpp:



Typedefs

- using `CommandHandler` = `std::function<std::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

<code>frame</code>	The Frame instance to encode.
--------------------	---

Returns

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

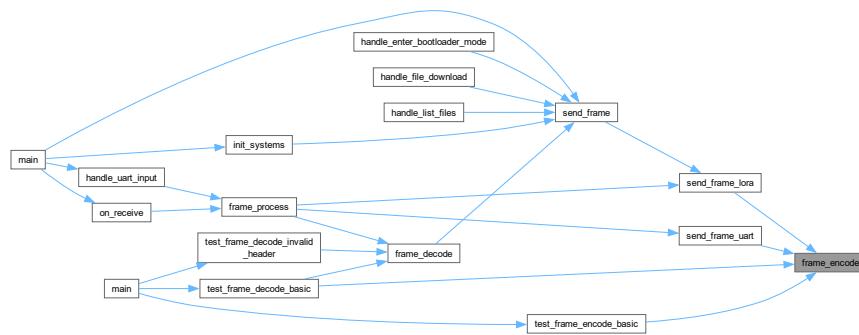
The encoded string includes the frame direction, operation type, group, command, value, and unit, all delimited by the DELIMITER character. The string is encapsulated by FRAME_BEGIN and FRAME_END.

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

<i>data</i>	The Frame data as a string.
-------------	---

Returns

The [Frame](#) instance.

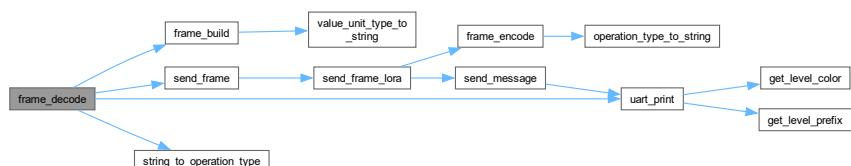
Exceptions

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

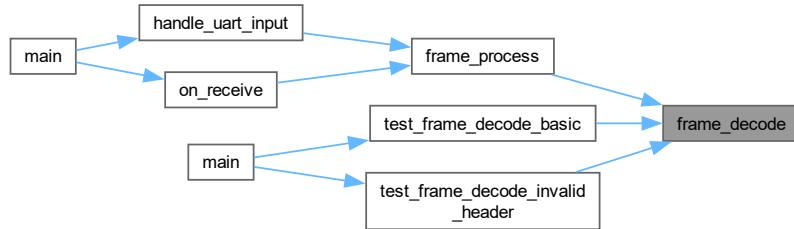
Parses the input string, extracting the frame direction, operation type, group, command, value, and unit. If an error occurs during parsing, an error message is printed to the UART, an error frame is built and sent, and the exception is re-thrown.

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.

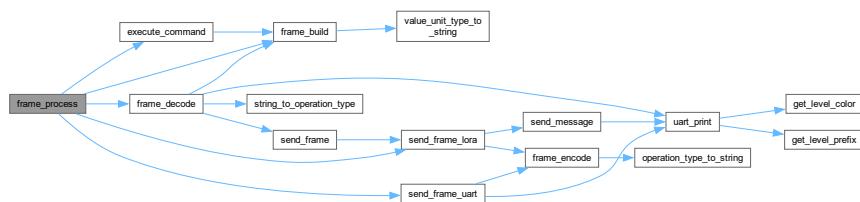
Parameters

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

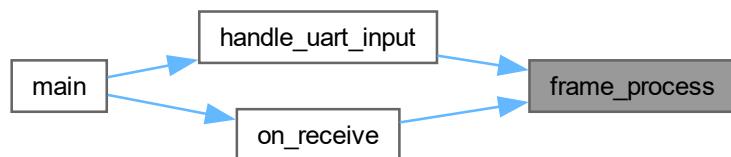
Decodes the frame data, extracts the command key, and executes the corresponding command. Sends the response frame. If an error occurs, an error frame is built and sent.

Definition at line 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

<i>result</i>	The execution result.
<i>group</i>	The group ID.
<i>command</i>	The command ID.
<i>value</i>	The value.
<i>unitType</i>	The value unit type.

Returns

The [Frame](#) instance.

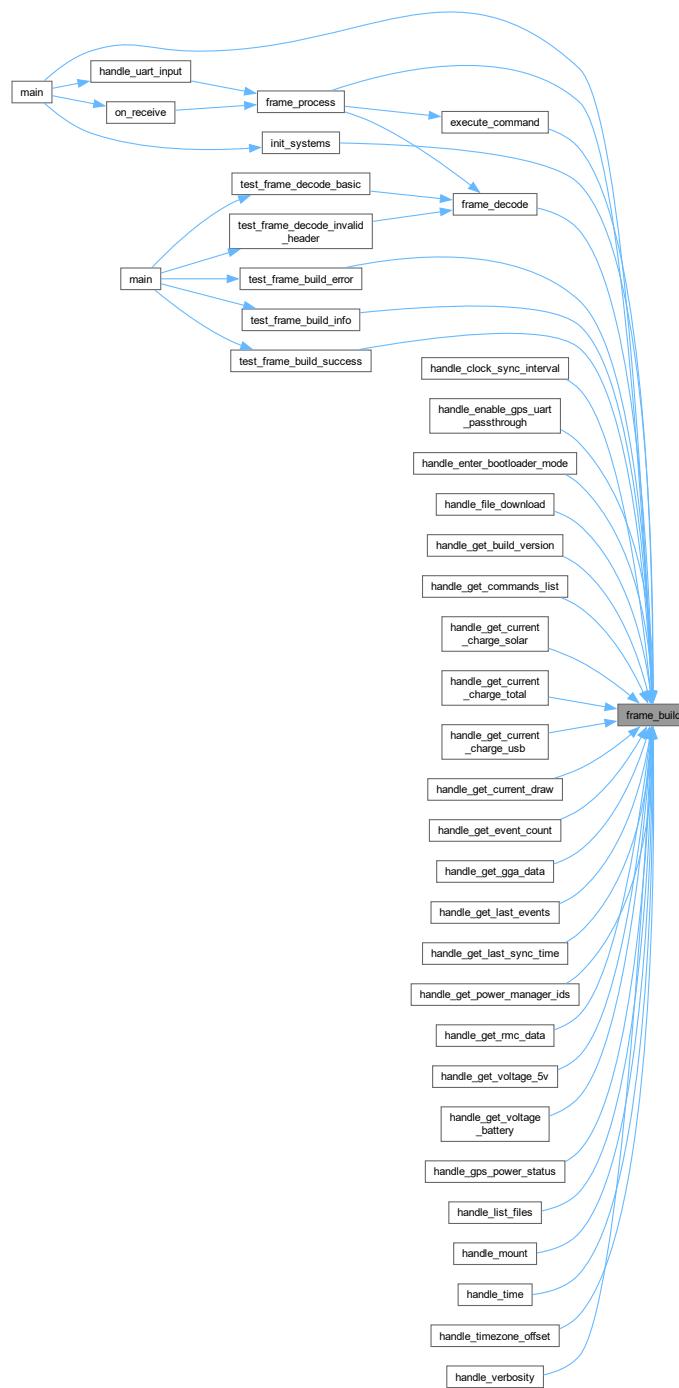
Constructs a [Frame](#) instance based on the provided parameters. The frame direction and operation type are set based on the execution result.

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.](#)

```

00001 #include "communication.h"
00002
00003 using CommandHandler = std::function<std::string(const std::string&, OperationType)>;
00004 extern std::map<uint32_t, CommandHandler> commandHandlers;
00005 extern volatile uint16_t eventRegister;
00006
00007
00008
00009 std::string frame_encode(const Frame& frame) {
00010     std::stringstream ss;
00011     ss << static_cast<int>(frame.direction) << DELIMITER
00012         << operation_type_to_string(frame.operationType) << DELIMITER
00013         << static_cast<int>(frame.group) << DELIMITER
00014         << static_cast<int>(frame.command) << DELIMITER
00015         << frame.value;
00016
00017     if (!frame.unit.empty()) {
00018         ss << DELIMITER << frame.unit;
00019     }
00020
00021     return FRAME_BEGIN + DELIMITER + ss.str() + DELIMITER + FRAME_END;
00022 }
00023
00024
00025
00026
00027
00028
00029
00030
00031
00032 }
00033
00034
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044 Frame frame_decode(const std::string& data) {
00045     try {
00046         Frame frame;
00047         std::stringstream ss(data);
00048         std::string token;
00049
00050         std::getline(ss, token, DELIMITER);
00051         if (token != FRAME_BEGIN)
00052             throw std::runtime_error("Invalid frame header");
00053         frame.header = token;
00054
00055         std::string decodedFrameData;
00056         while (std::getline(ss, token, DELIMITER)) {
00057             if (token == FRAME_END) break;
00058             decodedFrameData += token + DELIMITER;
00059         }
00060         if (!decodedFrameData.empty()) {
00061             decodedFrameData.pop_back();
00062         }
00063
00064         std::stringstream frameDataStream(decodedFrameData);
00065
00066         std::getline(frameDataStream, token, DELIMITER);
00067         frame.direction = std::stoi(token);
00068
00069         std::getline(frameDataStream, token, DELIMITER);
00070         frame.operationType = string_to_operation_type(token);
00071
00072         std::getline(frameDataStream, token, DELIMITER);
00073         frame.group = std::stoi(token);
00074
00075         std::getline(frameDataStream, token, DELIMITER);
00076         frame.command = std::stoi(token);
00077
00078         std::getline(frameDataStream, token, DELIMITER);
00079         frame.value = token;
00080
00081         std::getline(frameDataStream, token, DELIMITER);
00082         frame.unit = token;
00083
00084         return frame;
00085     } catch (const std::exception& e) {
00086         uart_print("Frame error: " + std::string(e.what()), VerbosityLevel::ERROR);
00087         Frame errorFrame = frame_build(ExecutionResult::ERROR, 0, 0, e.what());
00088         send_frame(errorFrame);
00089         throw;
00090     }
00091 }
00092
00093
00094
00095
00096
00097
00098
00099
00100 void frame_process(const std::string& data, Interface interface) {
00101     try {
00102         Frame frame = frame_decode(data);
00103         uint32_t commandKey = (static_cast<uint32_t>(frame.group) << 8) |
00104             static_cast<uint32_t>(frame.command);
00105         Frame responseFrame = execute_command(commandKey, frame.value, frame.operationType);
00106         // Send response through the same interface that received the command

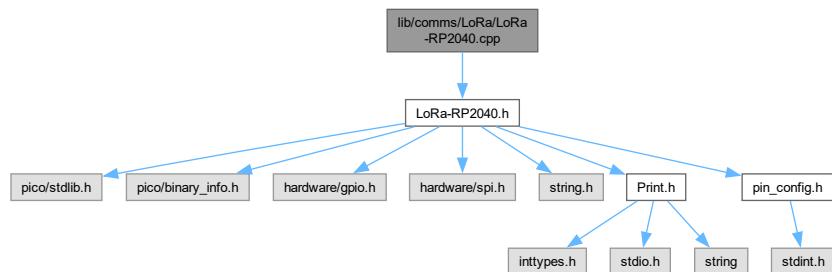
```

```

00108     if (interface == Interface::UART) {
00109         send_frame_uart(responseFrame);
00110     } else if (interface == Interface::LORA) {
00111         send_frame_lora(responseFrame);
00112     }
00113 } catch (const std::exception& e) {
00114     Frame errorFrame = frame_build(ExecutionResult::ERROR, 0, 0, e.what());
00115     // Send error through the same interface
00116     if (interface == Interface::UART) {
00117         send_frame_uart(errorFrame);
00118     } else if (interface == Interface::LORA) {
00119         send_frame_lora(errorFrame);
00120     }
00121 }
00122 }
00123
00135 Frame frame_build(ExecutionResult result, uint8_t group, uint8_t command,
00136                     const std::string& value, const ValueUnit unitType) {
00137     Frame frame;
00138     frame.header = FRAME_BEGIN;
00139     frame.footer = FRAME_END;
00140
00141     switch (result) {
00142         case ExecutionResult::SUCCESS:
00143             frame.direction = 1;
00144             frame.operationType = OperationType::ANS;
00145             frame.value = value;
00146             frame.unit = value_unit_type_to_string(unitType);
00147             break;
00148
00149         case ExecutionResult::ERROR:
00150             frame.direction = 1;
00151             frame.operationType = OperationType::ERR;
00152             frame.value = value;
00153             frame.unit = value_unit_type_to_string(ValueUnit::UNDEFINED);
00154             break;
00155
00156         case ExecutionResult::INFO:
00157             frame.direction = 1;
00158             frame.operationType = OperationType::INF;
00159             frame.value = value;
00160             frame.unit = value_unit_type_to_string(ValueUnit::UNDEFINED);
00161             break;
00162     }
00163
00164     frame.group = group;
00165     frame.command = command;
00166
00167     return frame;
00168 }
```

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.](#)

```

00001 #include "LoRa-RP2040.h"
00002 // registers
00003 #define REG_FIFO          0x00
00004 #define REG_OP_MODE        0x01
00005 #define REG_FRF_MSB         0x06
00006 #define REG_FRF_MID         0x07
00007 #define REG_FRF_LSB         0x08
00008 #define REG_PA_CONFIG       0x09
00009 #define IRQ_CAD_DETECTED_MASK 0x01
00010 #define REG_OCP            0x0b
00011 #define REG_LNA            0x0c
00012 #define IRQ_CAD_DONE_MASK   0x04
00013 #define REG_FIFO_ADDR_PTR   0x0d
00014 #define REG_FIFO_TX_BASE_ADDR 0x0e
00015 #define REG_FIFO_RX_BASE_ADDR 0x0f
00016 #define REG_FIFO_RX_CURRENT_ADDR 0x10
00017 #define REG_IRQ_FLAGS        0x12
00018 #define REG_RX_NB_BYTES      0x13
00019 #define REG_PKT_SNR_VALUE    0x19
00020 #define REG_PKT_RSSI_VALUE    0x1a
00021 #define REG_RSSI_VALUE        0x1b
00022 #define REG_MODEM_CONFIG_1     0x1d
00023 #define REG_MODEM_CONFIG_2     0x1e
00024 #define REG_PREAMBLE_MSB      0x20
00025 #define REG_PREAMBLE_LSB      0x21
00026 #define REG_PAYLOAD_LENGTH    0x22
00027 #define REG_MODEM_CONFIG_3     0x26
00028 #define REG_FREQ_ERROR_MSB     0x28
00029 #define REG_FREQ_ERROR_MID      0x29
00030 #define REG_FREQ_ERROR_LSB     0x2a
00031 #define REG_RSSI_WIDEBAND      0x2c
00032 #define REG_DETECTION_OPTIMIZE 0x31
00033 #define REG_INVERTIQ          0x33
00034 #define REG_DETECTION_THRESHOLD 0x37
00035 #define REG_SYNC_WORD          0x39
00036 #define REG_INVERTIQ2          0x3b
00037 #define REG_DIO_MAPPING_1       0x40
00038 #define REG_VERSION           0x42
00039 #define REG_PA_DAC            0x4d
00040 #define MODE_CAD              0x07
00041 // modes
00042 #define MODE_LONG_RANGE_MODE  0x80
00043 #define MODE_SLEEP             0x00
00044 #define MODE_STDBY            0x01
00045 #define MODE_TX               0x03
00046 #define MODE_RX_CONTINUOUS     0x05
00047 #define MODE_RX_SINGLE        0x06
00048
00049 // PA config
00050 #define PA_BOOST              0x80
00051
00052 // IRQ masks
00053 #define IRQ_TX_DONE_MASK      0x08
00054 #define IRQ_PAYLOAD_CRC_ERROR_MASK 0x20
00055 #define IRQ_RX_DONE_MASK       0x40
00056
00057 #define RF_MID_BAND_THRESHOLD  525E6
00058 #define RSSI_OFFSET_HF_PORT    157
00059 #define RSSI_OFFSET_LF_PORT    164
00060
00061 #define MAX_PKT_LENGTH         255
00062
00063 #if (ESP8266 || ESP32)
00064 #define ISR_PREFIX ICACHE_RAM_ATTR
00065 #else
00066 #define ISR_PREFIX
00067 #endif
00068
00069 LoRaClass::LoRaClass() :
00070     _spi(SPI_PORT),
00071     _ss(LORA_DEFAULT_SS_PIN), _reset(LORA_DEFAULT_RESET_PIN), _dio0(LORA_DEFAULT_DIO0_PIN),
00072     _frequency(0),
00073     _packetIndex(0),
00074     _implicitHeaderMode(0),
00075     _onReceive(NULL),
00076     _onCadDone(NULL),
00077     _onTxDone(NULL)
00078 {}
00079
00080 int LoRaClass::begin(long frequency)
00081 {
00082

```

```

00083 // setup pins
00084 gpio_init(_ss);
00085 gpio_set_dir(_ss, GPIO_OUT);
00086 // set SS high
00087 gpio_put(_ss, 1);
00088
00089 if (_reset != -1) {
00090     gpio_init(_reset);
00091     gpio_set_dir(_reset, GPIO_OUT);
00092
00093     // perform reset
00094     gpio_put(_reset, 0);
00095     sleep_ms(10);
00096     gpio_put(_reset, 1);
00097     sleep_ms(10);
00098 }
00099
00100
00101 // start SPI
00102 spi_init(SPI_PORT, LORA_DEFAULT_SPI_FREQUENCY);
00103 gpio_set_function(SX1278_MISO, GPIO_FUNC_SPI);
00104 gpio_set_function(SX1278_SCK, GPIO_FUNC_SPI);
00105 gpio_set_function(SX1278_MOSI, GPIO_FUNC_SPI);
00106
00107
00108 // Make the SPI pins available to picotool
00109 bi_decl(bi_3pins_with_func(SX1278_MISO, SX1278_SCK, SX1278_MOSI, GPIO_FUNC_SPI));
00110
00111 gpio_init(SX1278_CS);
00112 gpio_set_dir(SX1278_CS, GPIO_OUT);
00113 gpio_put(SX1278_CS, 1);
00114
00115 // Make the CS pin available to picotool
00116 bi_decl(bi_1pin_with_name(SX1278_CS, "SPI CS"));
00117
00118 // check version
00119 uint8_t version = readRegister(REG_VERSION);
00120 if (version != 0x12) {
00121     return 0;
00122 }
00123
00124 // put in sleep mode
00125 sleep();
00126
00127 // set frequency
00128 setFrequency(frequency);
00129
00130 // set base addresses
00131 writeRegister(REG_FIFO_TX_BASE_ADDR, 0);
00132 writeRegister(REG_FIFO_RX_BASE_ADDR, 0);
00133
00134 // set LNA boost
00135 writeRegister(REG_LNA, readRegister(REG_LNA) | 0x03);
00136
00137 // set auto AGC
00138 writeRegister(REG_MODEM_CONFIG_3, 0x04);
00139
00140 // set output power to 17 dBm
00141 setTxPower(17);
00142
00143 // put in standby mode
00144 idle();
00145
00146 return 1;
00147 }
00148
00149 void LoRaClass::end()
00150 {
00151     // put in sleep mode
00152     sleep();
00153
00154     // stop SPI
00155     spi_deinit(SPI_PORT);
00156 }
00157
00158 int LoRaClass::beginPacket(int implicitHeader)
00159 {
00160     if (isTransmitting()) {
00161         return 0;
00162     }
00163
00164     // put in standby mode
00165     idle();
00166
00167     if (implicitHeader) {
00168         implicitHeaderMode();
00169     } else {

```

```

00170     explicitHeaderMode();
00171 }
00172
00173 // reset FIFO address and payload length
00174 writeRegister(REG_FIFO_ADDR_PTR, 0);
00175 writeRegister(REG_PAYLOAD_LENGTH, 0);
00176
00177 return 1;
00178 }
00179
00180 int LoRaClass::endPacket(bool async)
00181 {
00182
00183 if ((async) && (_onTxDone)) {
00184     writeRegister(REG_DIO_MAPPING_1, 0x40); // DIO0 => TXDONE
00185 }
00186 // put in TX mode
00187 writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_TX);
00188
00189 if (!async) {
00190     // wait for TX done
00191     while ((readRegister(REG_IRQ_FLAGS) & IRQ_TX_DONE_MASK) == 0) {
00192         sleep_ms(0);
00193     }
00194     // clear IRQ's
00195     writeRegister(REG_IRQ_FLAGS, IRQ_TX_DONE_MASK);
00196 }
00197
00198 return 1;
00199 }
00200
00201 bool LoRaClass::isTransmitting()
00202 {
00203     if ((readRegister(REG_OP_MODE) & MODE_TX) == MODE_TX) {
00204         return true;
00205     }
00206
00207     if (readRegister(REG_IRQ_FLAGS) & IRQ_TX_DONE_MASK) {
00208         // clear IRQ's
00209         writeRegister(REG_IRQ_FLAGS, IRQ_TX_DONE_MASK);
00210     }
00211
00212     return false;
00213 }
00214
00215 int LoRaClass::parse_packet(int size)
00216 {
00217     int packetLength = 0;
00218
00219     int irqFlags = readRegister(REG_IRQ_FLAGS);
00220
00221     if (size > 0) {
00222         implicitHeaderMode();
00223         writeRegister(REG_PAYLOAD_LENGTH, size & 0xff);
00224     } else {
00225         explicitHeaderMode();
00226     }
00227
00228     // clear IRQ's
00229     writeRegister(REG_IRQ_FLAGS, irqFlags);
00230     writeRegister(REG_IRQ_FLAGS, irqFlags);
00231
00232     if ((irqFlags & IRQ_RX_DONE_MASK) && (irqFlags & IRQ_PAYLOAD_CRC_ERROR_MASK) == 0) {
00233         // received a packet
00234         _packetIndex = 0;
00235
00236         // read packet length
00237         if (_implicitHeaderMode) {
00238             packetLength = readRegister(REG_PAYLOAD_LENGTH);
00239         } else {
00240             packetLength = readRegister(REG_RX_NB_BYTES);
00241         }
00242
00243         // set FIFO address to current RX address
00244         writeRegister(REG_FIFO_ADDR_PTR, readRegister(REG_FIFO_RX_CURRENT_ADDR));
00245
00246         // put in standby mode
00247         idle();
00248     } else if (readRegister(REG_OP_MODE) != (MODE_LONG_RANGE_MODE | MODE_RX_SINGLE)) {
00249         // not currently in RX mode
00250
00251         // reset FIFO address
00252         writeRegister(REG_FIFO_ADDR_PTR, 0);
00253
00254         // put in single RX mode
00255         writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_RX_SINGLE);
00256     }

```

```
00257     return packetLength;
00258 }
00259
00260
00261 int LoRaClass::packetRssi()
00262 {
00263     return (readRegister(REG_PKT_RSSI_VALUE) - (_frequency < RF_MID_BAND_THRESHOLD ? RSSI_OFFSET_LF_PORT : RSSI_OFFSET_HF_PORT));
00264 }
00265
00266 float LoRaClass::packetSnr()
00267 {
00268     return ((int8_t)readRegister(REG_PKT_SNR_VALUE)) * 0.25;
00269 }
00270
00271 long LoRaClass::packetFrequencyError()
00272 {
00273     int32_t freqError = 0;
00274     freqError = static_cast<int32_t>(readRegister(REG_FREQ_ERROR_MSB) & 0x111);
00275     freqError <<= 8L;
00276     freqError += static_cast<int32_t>(readRegister(REG_FREQ_ERROR_MID));
00277     freqError <<= 8L;
00278     freqError += static_cast<int32_t>(readRegister(REG_FREQ_ERROR_LSB));
00279
00280     if (readRegister(REG_FREQ_ERROR_MSB) & 0x1000) { // Sign bit is on
00281         freqError -= 524288; // B1000'0000'0000'0000'0000
00282     }
00283
00284     const float fXtal = 32E6; // FXOSC: crystal oscillator (XTAL) frequency (2.5. Chip Specification, p. 14)
00285     const float fError = ((static_cast<float>(freqError) * (1L << 24)) / fXtal) * (getSignalBandwidth() / 500000.0f); // p. 37
00286
00287     return static_cast<long>(fError);
00288 }
00289
00290 int LoRaClass::rssи()
00291 {
00292     return (readRegister(REG_RSSI_VALUE) - (_frequency < RF_MID_BAND_THRESHOLD ? RSSI_OFFSET_LF_PORT : RSSI_OFFSET_HF_PORT));
00293 }
00294
00295 size_t LoRaClass::write(uint8_t byte)
00296 {
00297     return write(&byte, sizeof(byte));
00298 }
00299
00300 size_t LoRaClass::write(const uint8_t *buffer, size_t size)
00301 {
00302     int currentLength = readRegister(REG_PAYLOAD_LENGTH);
00303
00304     // check size
00305     if ((currentLength + size) > MAX_PKT_LENGTH) {
00306         size = MAX_PKT_LENGTH - currentLength;
00307     }
00308
00309     // write data
00310     for (size_t i = 0; i < size; i++) {
00311         writeRegister(REG_FIFO, buffer[i]);
00312     }
00313
00314     // update length
00315     writeRegister(REG_PAYLOAD_LENGTH, currentLength + size);
00316
00317     return size;
00318 }
00319
00320 int LoRaClass::available()
00321 {
00322     return (readRegister(REG_RX_NB_BYTES) - _packetIndex);
00323 }
00324
00325 int LoRaClass::read()
00326 {
00327     if (!available()) {
00328         return -1;
00329     }
00330
00331     _packetIndex++;
00332
00333     return readRegister(REG_FIFO);
00334 }
00335
00336 int LoRaClass::peek()
00337 {
00338     if (!available()) {
00339         return -1;
00340     }
```

```
00340 }
00341 // store current FIFO address
00342 int currentAddress = readRegister(REG_FIFO_ADDR_PTR);
00344
00345 // read
00346 uint8_t b = readRegister(REG_FIFO);
00347
00348 // restore FIFO address
00349 writeRegister(REG_FIFO_ADDR_PTR, currentAddress);
00350
00351 return b;
00352 }
00353
00354 void LoRaClass::flush()
00355 {
00356 }
00357
00358 void LoRaClass::onReceive(void(*callback)(int))
00359 {
00360     _onReceive = callback;
00361
00362     if (callback) {
00363         gpio_set_irq_enabled_with_callback(_dio0, GPIO_IRQ_EDGE_RISE, true, &LoRaClass::onDio0Rise);
00364     } else {
00365         gpio_set_irq_enabled(_dio0, GPIO_IRQ_EDGE_RISE, false);
00366     }
00367 }
00368
00369 void LoRaClass::onCadDone(void(*callback)(bool))
00370 {
00371     _onCadDone = callback;
00372
00373     if (callback) {
00374         gpio_set_irq_enabled_with_callback(_dio0, GPIO_IRQ_EDGE_RISE, true,
00375                                         &LoRaClass::onDio0Rise);
00376     } else {
00377         gpio_set_irq_enabled(_dio0, GPIO_IRQ_EDGE_RISE, false);
00378     }
00379 }
00380
00381 void LoRaClass::onTxDone(void (*callback)())
00382 {
00383     _onTxDone = callback;
00384
00385     if (callback) {
00386         gpio_set_irq_enabled_with_callback(_dio0, GPIO_IRQ_EDGE_RISE, true, &LoRaClass::onDio0Rise);
00387     } else {
00388         gpio_set_irq_enabled(_dio0, GPIO_IRQ_EDGE_RISE, false);
00389     }
00390 }
00391
00392 void LoRaClass::receive(int size)
00393 {
00394
00395     writeRegister(REG_DIO_MAPPING_1, 0x00); // DIO0 => RXDONE
00396
00397     if (size > 0) {
00398         implicitHeaderMode();
00399
00400         writeRegister(REG_PAYLOAD_LENGTH, size & 0xff);
00401     } else {
00402         explicitHeaderMode();
00403     }
00404
00405     writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_RX_CONTINUOUS);
00406 }
00407
00408 void LoRaClass::channelActivityDetection(void)
00409 {
00410     writeRegister(REG_DIO_MAPPING_1, 0x80); // DIO0 => CADDONE
00411     writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_CAD);
00412 }
00413
00414 void LoRaClass::idle()
00415 {
00416     writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_STDBY);
00417 }
00418
00419 void LoRaClass::sleep()
00420 {
00421     writeRegister(REG_OP_MODE, MODE_LONG_RANGE_MODE | MODE_SLEEP);
00422 }
00423
00424 void LoRaClass::setTxPower(int level, int outputPin)
00425 {
00426     if (PA_OUTPUT_RFO_PIN == outputPin) {
```

```

00427 // RFO
00428 if (level < 0) {
00429     level = 0;
00430 } else if (level > 14) {
00431     level = 14;
00432 }
00433
00434     writeRegister(REG_PA_CONFIG, 0x70 | level);
00435 } else {
00436 // PA BOOST
00437     if (level > 17) {
00438         if (level > 20) {
00439             level = 20;
00440         }
00441
00442 // subtract 3 from level, so 18 - 20 maps to 15 - 17
00443     level -= 3;
00444
00445 // High Power +20 dBm Operation (Semtech SX1276/77/78/79 5.4.3.)
00446     writeRegister(REG_PA_DAC, 0x87);
00447     setOCP(140);
00448 } else {
00449     if (level < 2) {
00450         level = 2;
00451     }
00452 //Default value PA_HF/LF or +17dBm
00453     writeRegister(REG_PA_DAC, 0x84);
00454     setOCP(100);
00455 }
00456
00457     writeRegister(REG_PA_CONFIG, PA_BOOST | (level - 2));
00458 }
00459 }
00460
00461 void LoRaClass::setFrequency(long frequency)
00462 {
00463     _frequency = frequency;
00464
00465     uint64_t frf = ((uint64_t)frequency << 19) / 32000000;
00466
00467     writeRegister(REG_FRF_MSB, (uint8_t)(frf >> 16));
00468     writeRegister(REG_FRF_MID, (uint8_t)(frf >> 8));
00469     writeRegister(REG_FRF_LSB, (uint8_t)(frf >> 0));
00470 }
00471
00472 int LoRaClass::getSpreadingFactor()
00473 {
00474     return readRegister(REG_MODEM_CONFIG_2) >> 4;
00475 }
00476
00477 void LoRaClass::setSpreadingFactor(int sf)
00478 {
00479     if (sf < 6) {
00480         sf = 6;
00481     } else if (sf > 12) {
00482         sf = 12;
00483     }
00484
00485     if (sf == 6) {
00486         writeRegister(REG_DETECTION_OPTIMIZE, 0xc5);
00487         writeRegister(REG_DETECTION_THRESHOLD, 0x0c);
00488     } else {
00489         writeRegister(REG_DETECTION_OPTIMIZE, 0xc3);
00490         writeRegister(REG_DETECTION_THRESHOLD, 0xa);
00491     }
00492
00493     writeRegister(REG_MODEM_CONFIG_2, (readRegister(REG_MODEM_CONFIG_2) & 0x0f) | ((sf << 4) & 0xf0));
00494     setLdoFlag();
00495 }
00496
00497 long LoRaClass::getSignalBandwidth()
00498 {
00499     uint8_t bw = (readRegister(REG_MODEM_CONFIG_1) >> 4);
00500
00501     switch (bw) {
00502     case 0: return 7.8E3;
00503     case 1: return 10.4E3;
00504     case 2: return 15.6E3;
00505     case 3: return 20.8E3;
00506     case 4: return 31.25E3;
00507     case 5: return 41.7E3;
00508     case 6: return 62.5E3;
00509     case 7: return 125E3;
00510     case 8: return 250E3;
00511     case 9: return 500E3;
00512     }
00513 }
```

```
00514     return -1;
00515 }
00516
00517 void LoRaClass::setSignalBandwidth(long sbw)
00518 {
00519     int bw;
00520
00521     if (sbw <= 7.8E3) {
00522         bw = 0;
00523     } else if (sbw <= 10.4E3) {
00524         bw = 1;
00525     } else if (sbw <= 15.6E3) {
00526         bw = 2;
00527     } else if (sbw <= 20.8E3) {
00528         bw = 3;
00529     } else if (sbw <= 31.25E3) {
00530         bw = 4;
00531     } else if (sbw <= 41.7E3) {
00532         bw = 5;
00533     } else if (sbw <= 62.5E3) {
00534         bw = 6;
00535     } else if (sbw <= 125E3) {
00536         bw = 7;
00537     } else if (sbw <= 250E3) {
00538         bw = 8;
00539     } else /*if (sbw <= 250E3)*/ {
00540         bw = 9;
00541     }
00542
00543     writeRegister(REG_MODEM_CONFIG_1, (readRegister(REG_MODEM_CONFIG_1) & 0x0f) | (bw << 4));
00544     setLdoFlag();
00545 }
00546
00547 void LoRaClass::setLdoFlag()
00548 {
00549     long symbolDuration = 1000 / ( getSignalBandwidth() / (1L << getSpreadingFactor()) );
00550
00551     bool ldoOn = symbolDuration > 16;
00552
00553     uint8_t config3 = readRegister(REG_MODEM_CONFIG_3);
00554
00555     config3 = ldoOn ? config3 | (1 << 3) : config3 & ~ (1 << 3);
00556
00557     writeRegister(REG_MODEM_CONFIG_3, config3);
00558 }
00559
00560 void LoRaClass::setCodingRate4(int denominator)
00561 {
00562     if (denominator < 5) {
00563         denominator = 5;
00564     } else if (denominator > 8) {
00565         denominator = 8;
00566     }
00567
00568     int cr = denominator - 4;
00569
00570     writeRegister(REG_MODEM_CONFIG_1, (readRegister(REG_MODEM_CONFIG_1) & 0xf1) | (cr << 1));
00571 }
00572
00573 void LoRaClass::setPreambleLength(long length)
00574 {
00575     writeRegister(REG_PREAMBLE_MSB, (uint8_t)(length >> 8));
00576     writeRegister(REG_PREAMBLE_LSB, (uint8_t)(length >> 0));
00577 }
00578
00579 void LoRaClass::setSyncWord(int sw)
00580 {
00581     writeRegister(REG_SYNC_WORD, sw);
00582 }
00583
00584 void LoRaClass::enableCrc()
00585 {
00586     writeRegister(REG_MODEM_CONFIG_2, readRegister(REG_MODEM_CONFIG_2) | 0x04);
00587 }
00588
00589 void LoRaClass::disableCrc()
00590 {
00591     writeRegister(REG_MODEM_CONFIG_2, readRegister(REG_MODEM_CONFIG_2) & 0xfb);
00592 }
00593
00594 void LoRaClass::enableInvertIQ()
00595 {
00596     writeRegister(REG_INVERTIQ, 0x66);
00597     writeRegister(REG_INVERTIQ2, 0x19);
00598 }
00599
00600 void LoRaClass::disableInvertIQ()
```

```

00601 {
00602     writeRegister(REG_INVERTIQ, 0x27);
00603     writeRegister(REG_INVERTIQ2, 0x1d);
00604 }
00605
00606 void LoRaClass::setOCP(uint8_t mA)
00607 {
00608     uint8_t ocpTrim = 27;
00609
00610     if (mA <= 120) {
00611         ocpTrim = (mA - 45) / 5;
00612     } else if (mA <= 240) {
00613         ocpTrim = (mA + 30) / 10;
00614     }
00615
00616     writeRegister(REG_OCP, 0x20 | (0x1F & ocpTrim));
00617 }
00618
00619 void LoRaClass::setGain(uint8_t gain)
00620 {
00621     // check allowed range
00622     if (gain > 6) {
00623         gain = 6;
00624     }
00625
00626     // set to standby
00627     idle();
00628
00629     // set gain
00630     if (gain == 0) {
00631         // if gain = 0, enable AGC
00632         writeRegister(REG_MODEM_CONFIG_3, 0x04);
00633     } else {
00634         // disable AGC
00635         writeRegister(REG_MODEM_CONFIG_3, 0x00);
00636
00637         // clear Gain and set LNA boost
00638         writeRegister(REG_LNA, 0x03);
00639
00640         // set gain
00641         writeRegister(REG_LNA, readRegister(REG_LNA) | (gain << 5));
00642     }
00643 }
00644
00645 uint8_t LoRaClass::random()
00646 {
00647     return readRegister(REG_RSSI_WIDEBAND);
00648 }
00649
00650 void LoRaClass::set_pins(int ss, int reset, int dio0)
00651 {
00652     _ss = ss;
00653     _reset = reset;
00654     _dio0 = dio0;
00655 }
00656
00657 void LoRaClass::setSPI(spi_inst_t& spi)
00658 {
00659     _spi = &spi;
00660 }
00661
00662 void LoRaClass::setSPIFrequency(uint32_t frequency)
00663 {
00664     spi_set_baudrate(SPI_PORT, frequency);
00665 }
00666
00667 void LoRaClass::dumpRegisters()
00668 {
00669     for (int i = 0; i < 128; i++) {
00670         printf("0x%02x: 0x%02x\n", i, readRegister(i));
00671     }
00672 }
00673
00674 void LoRaClass::explicitHeaderMode()
00675 {
00676     _implicitHeaderMode = 0;
00677
00678     writeRegister(REG_MODEM_CONFIG_1, readRegister(REG_MODEM_CONFIG_1) & 0xfe);
00679 }
00680
00681 void LoRaClass::implicitHeaderMode()
00682 {
00683     _implicitHeaderMode = 1;
00684
00685     writeRegister(REG_MODEM_CONFIG_1, readRegister(REG_MODEM_CONFIG_1) | 0x01);
00686 }
00687

```

```

00688 void LoRaClass::handleDio0Rise()
00689 {
00690     int irqFlags = readRegister(REG_IRQ_FLAGS);
00691
00692     // clear IRQ's
00693     writeRegister(REG_IRQ_FLAGS, irqFlags);
00694     writeRegister(REG_IRQ_FLAGS, irqFlags);
00695
00696     if ((irqFlags & IRQ_CAD_DONE_MASK) != 0) {
00697         if (_onCadDone) {
00698             _onCadDone((irqFlags & IRQ_CAD_DETECTED_MASK) != 0);
00699         }
00700     } else if ((irqFlags & IRQ_PAYLOAD_CRC_ERROR_MASK) == 0) {
00701
00702         if ((irqFlags & IRQ_RX_DONE_MASK) != 0) {
00703             // received a packet
00704             _packetIndex = 0;
00705
00706             // read packet length
00707             int packetLength = _implicitHeaderMode ? readRegister(REG_PAYLOAD_LENGTH) :
00708                 readRegister(REG_RX_NB_BYTES);
00709
00710             // set FIFO address to current RX address
00711             writeRegister(REG_FIFO_ADDR_PTR, readRegister(REG_FIFO_RX_CURRENT_ADDR));
00712
00713             if (_onReceive) {
00714                 _onReceive(packetLength);
00715             }
00716         } else if ((irqFlags & IRQ_TX_DONE_MASK) != 0) {
00717             if (_onTxDone) {
00718                 _onTxDone();
00719             }
00720         }
00721     }
00722
00723 uint8_t LoRaClass::readRegister(uint8_t address)
00724 {
00725     return singleTransfer(address & 0x7f, 0x00);
00726 }
00727
00728 void LoRaClass::writeRegister(uint8_t address, uint8_t value)
00729 {
00730     singleTransfer(address | 0x80, value);
00731 }
00732
00733 uint8_t LoRaClass::singleTransfer(uint8_t address, uint8_t value)
00734 {
00735     uint8_t response;
00736
00737     gpio_put(_ss, 0);
00738
00739     spi_write_blocking(SPI_PORT, &address, 1);
00740     spi_write_read_blocking(SPI_PORT, &value, &response, 1);
00741
00742     gpio_put(_ss, 1);
00743
00744     return response;
00745 }
00746
00747 void LoRaClass::onDio0Rise(uint gpio, uint32_t events)
00748 {
00749     gpio_acknowledge_irq(gpio, events);
00750     LoRa.handleDio0Rise();
00751 }
00752
00753 LoRaClass LoRa;

```

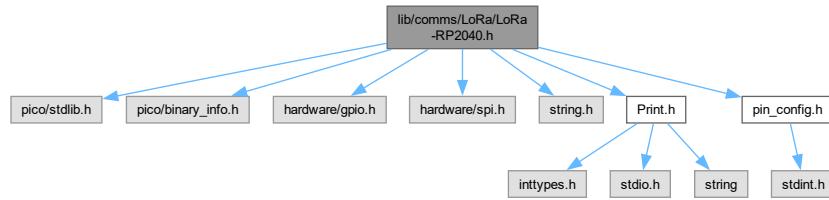
8.37 lib/comms/LoRa/LoRa-RP2040.h File Reference

```

#include "pico/stl.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.](#)

```

00001 #ifndef LORA_H
00002 #define LORA_H
00003
00004 #include "pico/stdlib.h"
00005 #include "pico/binary_info.h"
00006 #include "hardware/gpio.h"
00007 #include "hardware/spi.h"
00008 #include "string.h"
00009 #include "Print.h"
00010 #include "pin_config.h"
00011
00012 #endif
00013
00014 static void __empty();
00015
00016 //class LoRaClass : public Stream {
00017 class LoRaClass : public Print {
00018 public:
00019     LoRaClass();
00020
00021     int begin(long frequency);
00022     void end();
00023
00024     int beginPacket(int implicitHeader = false);
00025     int endPacket(bool async = false);
00026
00027     int parse_packet(int size = 0);
00028     int packetRssi();
00029     float packetSnr();
00030     long packetFrequencyError();
00031
00032     int rssi();
00033
00034     // from Print
00035     virtual size_t write(uint8_t byte);
00036     virtual size_t write(const uint8_t *buffer, size_t size);
00037
00038     // from Stream
00039     virtual int available();
00040     virtual int read();
00041     virtual int peek();
00042     virtual void flush();
00043
00044     void onCadDone(void (*callback)(bool));
00045     void onReceive(void (*callback)(int));
00046     void onTxDone(void (*callback)());
00047
00048     void receive(int size = 0);
00049     void channelActivityDetection(void);
00050
00051     void idle();
00052     void sleep();
00053
00054     // size_t print(const char* c);
00055
00056     void setTxPower(int level, int outputPin = PA_OUTPUT_PA_BOOST_PIN);
00057     void setFrequency(long frequency);
00058     void setSpreadingFactor(int sf);
00059     void setSignalBandwidth(long sbw);
00060     void setCodingRate4(int denominator);
00061     void setPreambleLength(long length);
00062     void setSyncWord(int sw);
00063     void enableCrc();
00064     void disableCrc();
00065     void enableInvertIQ();
00066     void disableInvertIQ();
00067
00068     void setOCP(uint8_t mA); // Over Current Protection control
00069
00070     void setGain(uint8_t gain); // Set LNA gain
00071
00072     // deprecated
00073     void crc() { enableCrc(); }
00074     void noCrc() { disableCrc(); }
00075
00076     uint8_t random();
00077
00078     void set_pins(int ss = LORA_DEFAULT_SS_PIN, int reset = LORA_DEFAULT_RESET_PIN, int dio0 =
00079     LORA_DEFAULT_DIO0_PIN);
00080     void setSPI(spi_inst_t &spi);
00081     void setSPIFrequency(uint32_t frequency);
00081

```

```

00082     void dumpRegisters();
00083
00084 private:
00085     void explicitHeaderMode();
00086     void implicitHeaderMode();
00087
00088     void handleDio0Rise();
00089     bool isTransmitting();
00090
00091     int getSpreadingFactor();
00092     long getSignalBandwidth();
00093
00094     void setLdoFlag();
00095
00096     uint8_t readRegister(uint8_t address);
00097     void writeRegister(uint8_t address, uint8_t value);
00098     uint8_t singleTransfer(uint8_t address, uint8_t value);
00099
00100    static void onDio0Rise(uint, uint32_t);
00101
00102 private:
00103    // SPISettings _spiSettings;
00104    spi_inst_t *_spi;
00105    int _ss;
00106    int _reset;
00107    int _dio0;
00108    long _frequency;
00109    int _packetIndex;
00110    int _implicitHeaderMode;
00111    void (*_onReceive)(int);
00112    void (*_onCadDone)(bool);
00113    void (*_onTxDone)();
00114 };
00115
00116 extern LoRaClass LoRa;

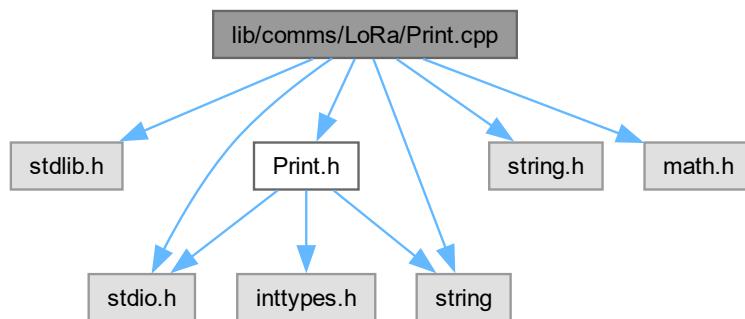
```

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.](#)

```
00001 /*
00002  Copyright (c) 2014 Arduino. 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 #include <stdlib.h>
00020 #include <stdio.h>
00021 #include <string.h>
00022 #include <string>
00023 #include <math.h>
00024
00025 #include "Print.h"
00026
00027 using std::string;
00028 // Public Methods /////////////////////////////////
00029
00030 /* default implementation: may be overridden */
00031 size_t Print::write(const uint8_t *buffer, size_t size)
00032 {
00033     size_t n = 0;
00034     while (size--) {
00035         if (write(buffer++)) n++;
00036         else break;
00037     }
00038     return n;
00039 }
00040
00041 size_t Print::print(string str)
00042 {
00043     return write(str.c_str(), str.size());
00044 }
00045
00046 // size_t Print::print(const char str[])
00047 //
00048 //    return write(str);
00049 //
00050
00051 size_t Print::print(char c)
00052 {
00053     return write(c);
00054 }
00055
00056 size_t Print::print(const char* c){
00057     return write(c);
00058 }
00059
00060
00061 size_t Print::print(unsigned char b, int base)
00062 {
00063     return print((unsigned long) b, base);
00064 }
00065
00066 size_t Print::print(int n, int base)
00067 {
00068     return print((long) n, base);
00069 }
00070
00071 size_t Print::print(unsigned int n, int base)
00072 {
00073     return print((unsigned long) n, base);
00074 }
00075
00076 size_t Print::print(long n, int base)
00077 {
00078     if (base == 0) {
00079         return write(n);
00080     } else if (base == 10) {
00081         if (n < 0) {
00082             int t = print('-');
00083             n = -n;
00084             return printNumber(n, 10) + t;
00085         }
00086         return printNumber(n, 10);
00087     } else {
```

```
00088     return printNumber(n, base);
00089 }
00090 }
00091
00092 size_t Print::print(unsigned long n, int base)
00093 {
00094     if (base == 0) return write(n);
00095     else return printNumber(n, base);
00096 }
00097
00098 size_t Print::print(long long n, int base)
00099 {
00100     if (base == 0) {
00101         return write(n);
00102     } else if (base == 10) {
00103         if (n < 0) {
00104             int t = print('-');
00105             n = -n;
00106             return printULLNumber(n, 10) + t;
00107         }
00108         return printULLNumber(n, 10);
00109     } else {
00110         return printULLNumber(n, base);
00111     }
00112 }
00113
00114 size_t Print::print(unsigned long long n, int base)
00115 {
00116     if (base == 0) return write(n);
00117     else return printULLNumber(n, base);
00118 }
00119
00120 size_t Print::print(double n, int digits)
00121 {
00122     return printFloat(n, digits);
00123 }
00124
00125 size_t Print::println(void)
00126 {
00127     return write("\r\n");
00128 }
00129
00130 size_t Print::println(const char c[])
00131 {
00132     size_t n = print(c);
00133     n += println();
00134     return n;
00135 }
00136
00137 size_t Print::println(char c)
00138 {
00139     size_t n = print(c);
00140     n += println();
00141     return n;
00142 }
00143
00144 size_t Print::println(unsigned char b, int base)
00145 {
00146     size_t n = print(b, base);
00147     n += println();
00148     return n;
00149 }
00150
00151 size_t Print::println(int num, int base)
00152 {
00153     size_t n = print(num, base);
00154     n += println();
00155     return n;
00156 }
00157
00158 size_t Print::println(unsigned int num, int base)
00159 {
00160     size_t n = print(num, base);
00161     n += println();
00162     return n;
00163 }
00164
00165 size_t Print::println(long num, int base)
00166 {
00167     size_t n = print(num, base);
00168     n += println();
00169     return n;
00170 }
00171
00172 size_t Print::println(unsigned long num, int base)
00173 {
00174     size_t n = print(num, base);
```

```

00175     n += println();
00176     return n;
00177 }
00178
00179 size_t Print::println(long long num, int base)
00180 {
00181     size_t n = print(num, base);
00182     n += println();
00183     return n;
00184 }
00185
00186 size_t Print::println(unsigned long long num, int base)
00187 {
00188     size_t n = print(num, base);
00189     n += println();
00190     return n;
00191 }
00192
00193 size_t Print::println(double num, int digits)
00194 {
00195     size_t n = print(num, digits);
00196     n += println();
00197     return n;
00198 }
00199
00200
00201 // Private Methods /////////////////////////////////
00202
00203 size_t Print::printNumber(unsigned long n, uint8_t base)
00204 {
00205     char buf[8 * sizeof(long) + 1]; // Assumes 8-bit chars plus zero byte.
00206     char *str = &buf[sizeof(buf) - 1];
00207
00208     *str = '\0';
00209
00210     // prevent crash if called with base == 1
00211     if (base < 2) base = 10;
00212
00213     do {
00214         char c = n % base;
00215         n /= base;
00216
00217         *--str = c < 10 ? c + '0' : c + 'A' - 10;
00218     } while(n);
00219
00220     return write(str);
00221 }
00222
00223 // REFERENCE IMPLEMENTATION FOR ULL
00224 // size_t Print::printULLNumber(unsigned long long n, uint8_t base)
00225 // {
00226     // // if limited to base 10 and 16 the bufsize can be smaller
00227     // char buf[65];
00228     // char *str = &buf[64];
00229
00230     // *str = '\0';
00231
00232     // // prevent crash if called with base == 1
00233     // if (base < 2) base = 10;
00234
00235     // do {
00236         // unsigned long long t = n / base;
00237         // char c = n - t * base; // faster than c = n%base;
00238         // n = t;
00239         // *--str = c < 10 ? c + '0' : c + 'A' - 10;
00240     // } while(n);
00241
00242     // return write(str);
00243 // }
00244
00245 // FAST IMPLEMENTATION FOR ULL
00246 size_t Print::printULLNumber(unsigned long long n64, uint8_t base)
00247 {
00248     // if limited to base 10 and 16 the bufsize can be 20
00249     char buf[64];
00250     uint8_t i = 0;
00251     uint8_t innerLoops = 0;
00252
00253     // prevent crash if called with base == 1
00254     if (base < 2) base = 10;
00255
00256     // process chunks that fit in "16 bit math".
00257     uint16_t top = 0xFFFF / base;
00258     uint16_t th16 = 1;
00259     while (th16 < top)
00260     {
00261         th16 *= base;

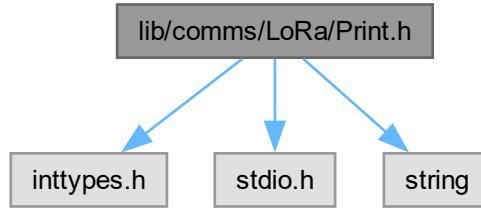
```

```

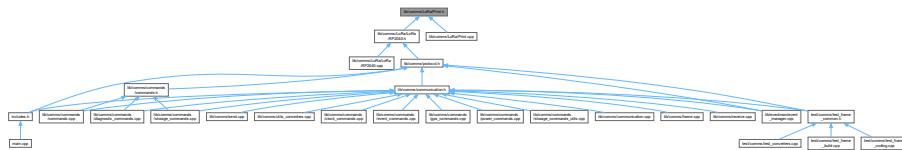
00262     innerLoops++;
00263 }
00264
00265 while (n64 > th16)
00266 {
00267     // 64 bit math part
00268     uint64_t q = n64 / th16;
00269     uint16_t r = n64 - q*th16;
00270     n64 = q;
00271
00272     // 16 bit math loop to do remainder. (note buffer is filled reverse)
00273     for (uint8_t j=0; j < innerLoops; j++)
00274     {
00275         uint16_t qq = r/base;
00276         buf[i++] = r - qq*base;
00277         r = qq;
00278     }
00279 }
00280
00281 uint16_t n16 = n64;
00282 while (n16 > 0)
00283 {
00284     uint16_t qq = n16/base;
00285     buf[i++] = n16 - qq*base;
00286     n16 = qq;
00287 }
00288
00289 size_t bytes = i;
00290 for (; i > 0; i--)
00291     write((char) (buf[i - 1] < 10 ?
00292      '0' + buf[i - 1] :
00293      'A' + buf[i - 1] - 10));
00294
00295     return bytes;
00296 }
00297
00298 size_t Print::printFloat(double number, int digits)
00299 {
00300     if (digits < 0)
00301         digits = 2;
00302
00303     size_t n = 0;
00304
00305     if (isnan(number)) return print("nan");
00306     if (isinf(number)) return print("inf");
00307     if (number > 4294967040.0) return print ("ovf"); // constant determined empirically
00308     if (number <-4294967040.0) return print ("ovf"); // constant determined empirically
00309
00310     // Handle negative numbers
00311     if (number < 0.0)
00312     {
00313         n += print('-');
00314         number = -number;
00315     }
00316
00317     // Round correctly so that print(1.999, 2) prints as "2.00"
00318     double rounding = 0.5;
00319     for (uint8_t i=0; i<digits; ++i)
00320         rounding /= 10.0;
00321
00322     number += rounding;
00323
00324     // Extract the integer part of the number and print it
00325     unsigned long int_part = (unsigned long)number;
00326     double remainder = number - (double)int_part;
00327     n += print(int_part);
00328
00329     // Print the decimal point, but only if there are digits beyond
00330     if (digits > 0) {
00331         n += print(".");
00332     }
00333
00334     // Extract digits from the remainder one at a time
00335     while (digits-- > 0)
00336     {
00337         remainder *= 10.0;
00338         unsigned int toPrint = (unsigned int)remainder;
00339         n += print(toPrint);
00340         remainder -= toPrint;
00341     }
00342
00343     return n;
00344 }
```

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         int getWriteError() { return write_error; }
```

```

00047     void clearWriteError() { setWriteError(0); }
00048
00049     virtual size_t write(uint8_t) = 0;
00050     size_t write(const char *str) {
00051         if (str == NULL) return 0;
00052         return write((const uint8_t *)str, strlen(str));
00053     }
00054     virtual size_t write(const uint8_t *buffer, size_t size);
00055     size_t write(const char *buffer, size_t size) {
00056         return write((const uint8_t *)buffer, size);
00057     }
00058
00059     // default to zero, meaning "a single write may block"
00060     // should be overridden by subclasses with buffering
00061     virtual int availableForWrite() { return 0; }
00062
00063     // size_t print(const char[]);
00064     size_t print(char);
00065     size_t print(const char*);
00066     size_t print(string c);
00067     size_t print(unsigned char, int = DEC);
00068     size_t print(int, int = DEC);
00069     size_t print(unsigned int, int = DEC);
00070     size_t print(long, int = DEC);
00071     size_t print(unsigned long, int = DEC);
00072     size_t print(long long, int = DEC);
00073     size_t print(unsigned long long, int = DEC);
00074     size_t print(double, int = 2);
00075
00076     size_t println(const char[]);
00077     size_t println(char);
00078     size_t println(unsigned char, int = DEC);
00079     size_t println(int, int = DEC);
00080     size_t println(unsigned int, int = DEC);
00081     size_t println(long, int = DEC);
00082     size_t println(unsigned long, int = DEC);
00083     size_t println(long long, int = DEC);
00084     size_t println(unsigned long long, int = DEC);
00085     size_t println(double, int = 2);
00086     size_t println(void);
00087
00088     virtual void flush() { /* Empty implementation for backward compatibility */ }
00089 };
00090

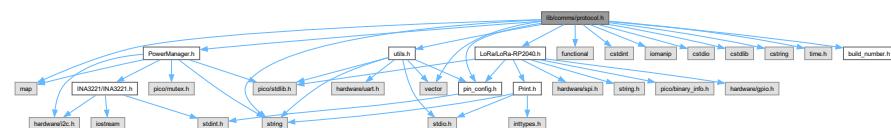
```

8.43 lib/comms/protocol.h File Reference

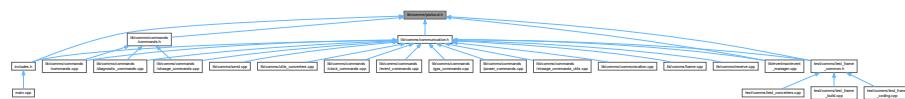
```

#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_UNNECESSARY` }
 - 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_UNNECESSARY	

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

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

Returns

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

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

Here is the caller graph for this function:



8.43.2.2 `operation_type_to_string()`

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

Converts an `OperationType` to a string.

Parameters

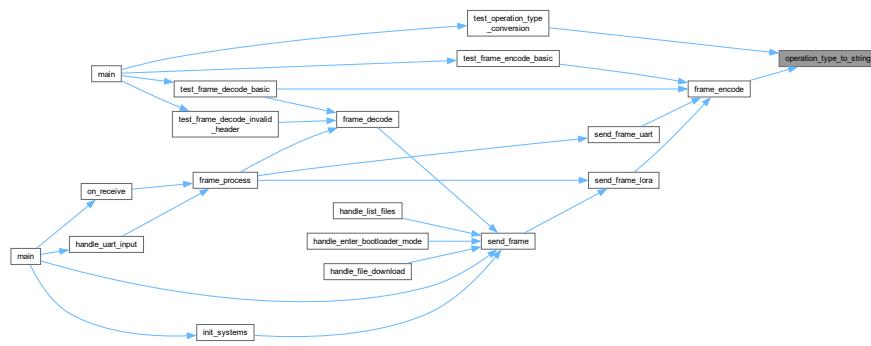
<code>type</code>	The <code>OperationType</code> to convert.
-------------------	--

Returns

The string representation of the `OperationType`.

Definition at line 50 of file `utils_converters.cpp`.

Here is the caller graph for this function:

**8.43.2.3 string_to_operation_type()**

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

Converts a string to an `OperationType`.

Parameters

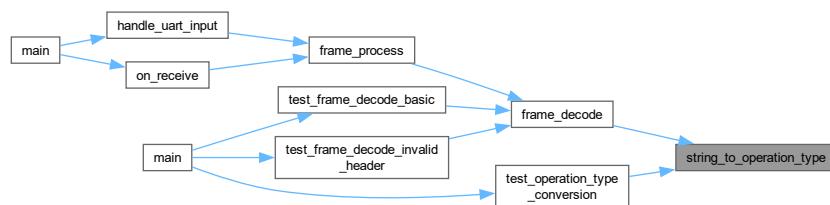
<code>str</code>	The string to convert.
------------------	------------------------

Returns

The `OperationType` corresponding to the string. Defaults to GET if the string is not recognized.

Definition at line 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

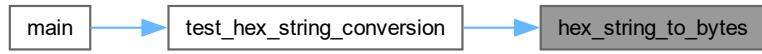
<i>hexString</i>	The hex string to convert.
------------------	----------------------------

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

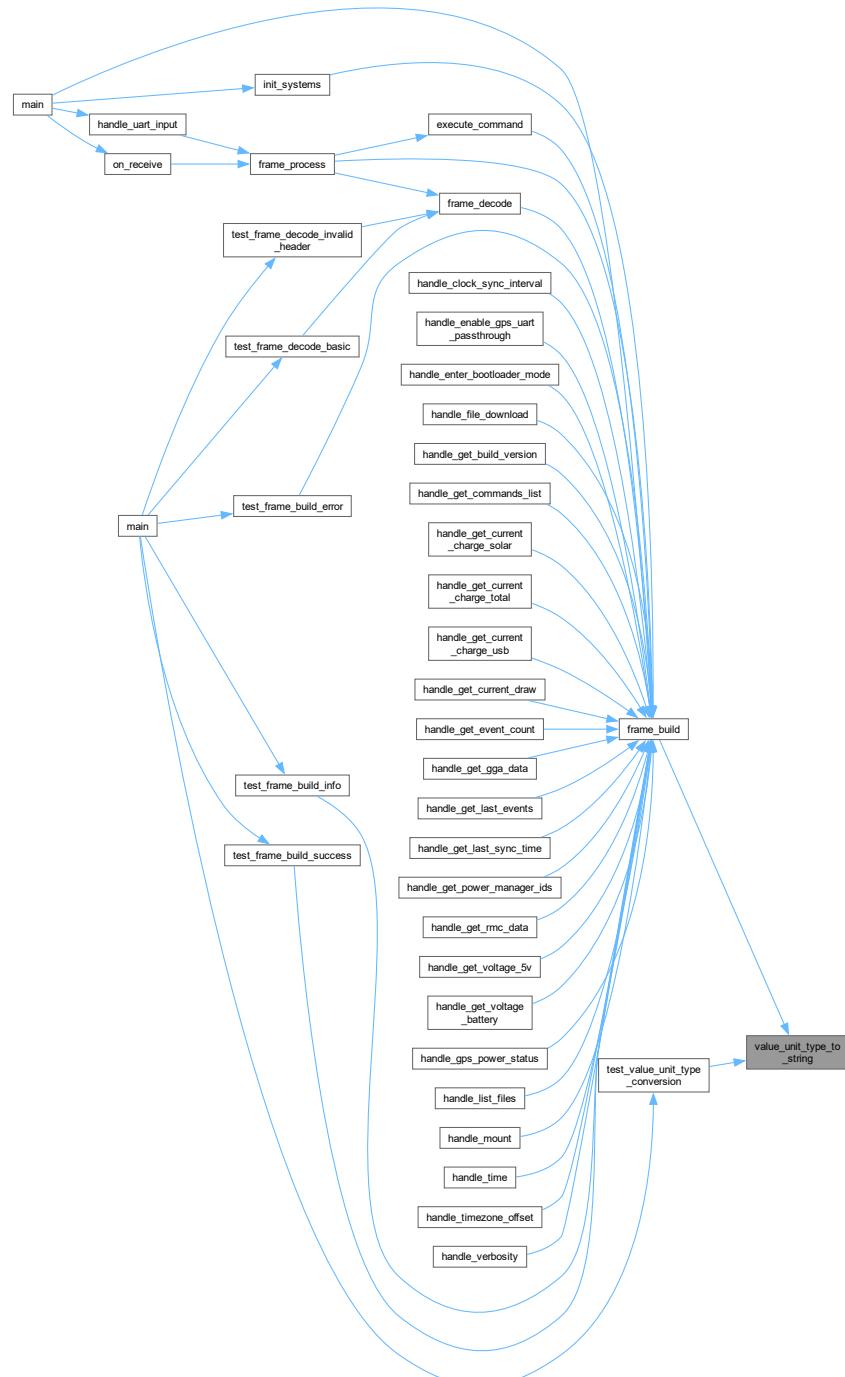
<i>unit</i>	The ValueUnit to convert.
-------------	---

Returns

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

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

Here is the caller graph for this function:



8.43.3 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,
```

```

00043     WRITE_ONLY,
00044     READ_WRITE
00045 };
00046
00047 enum class ValueUnit {
00048     UNDEFINED,
00049     SECOND,
00050     VOLT,
00051     BOOL,
00052     DATETIME,
00053     TEXT,
00054     MILLIAMP,
00055 };
00056
00057 enum class ExceptionType {
00058     NONE,
00059     NOT_ALLOWED,
00060     INVALID_PARAM,
00061     INVALID_OPERATION,
00062     PARAM_UNNECESSARY
00063 };
00064
00065 enum class Interface {
00066     UART,
00067     LORA
00068 };
00069
00070 struct Frame {
00071     std::string header;           // Start marker
00072     uint8_t direction;           // 0 = ground->sat, 1 = sat->ground
00073     OperationType operationType; // Group ID
00074     uint8_t group;               // Command ID within group
00075     uint8_t command;             // Payload value
00076     std::string value;           // Payload unit
00077     std::string unit;            // End marker
00078     std::string footer;
00079 };
00080
00081 std::string exception_type_to_string(ExceptionType type);
00082 std::string operation_type_to_string(OperationType type);
00083 OperationType string_to_operation_type(const std::string& str);
00084 std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString);
00085 std::string value_unit_type_to_string(ValueUnit unit);
00086
00087 #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.

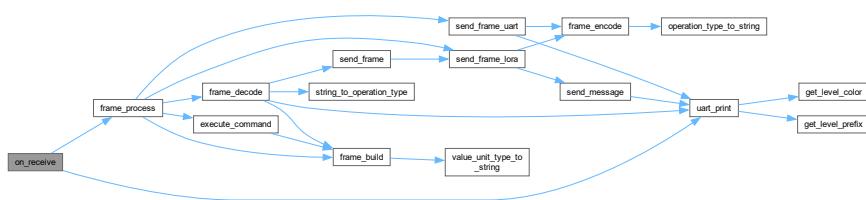
Parameters

<i>packetSize</i>	The size of the received packet.
-------------------	----------------------------------

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

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.45.2.2 handle_uart_input()

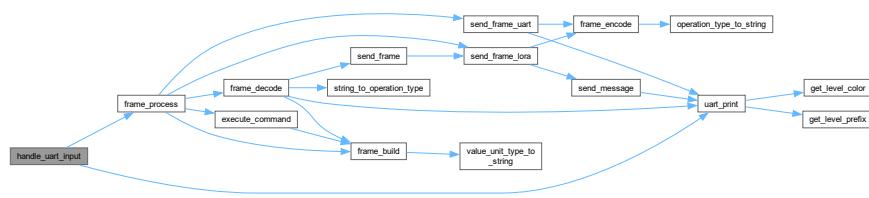
```
void handle_uart_input ()
```

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.](#)

```

00001 #include "communication.h"
00002
00003
00008
00015 void on_receive(int packetSize) {
00016     if (packetSize == 0) return;
00017
00018     uint8_t buffer[256];
00019     int bytesRead = 0;
00020
00021     while (LoRa.available() && bytesRead < packetSize) {
00022         buffer[bytesRead++] = LoRa.read();
00023     }
00024
00025     // Extract LoRa metadata
00026     uint8_t receivedDestination = buffer[0];
00027     uint8_t receivedLocalAddress = buffer[1];
00028
00029     // Validate metadata (optional, for security)
00030     if (receivedDestination != lora_address_local) {
00031         uart_print("Error: Destination address mismatch!", VerbosityLevel::ERROR);
00032         return;
00033     }
00034 }
```

```

00035     if (receivedLocalAddress != lora_address_remote) {
00036         uart_print("Error: Local address mismatch!", VerbosityLevel::ERROR);
00037         return;
00038     }
00039
00040     // Find the starting index of the actual frame data
00041     int startIndex = 2; // Start after the metadata
00042
00043     // Extract the frame data
00044     std::string received = std::string(reinterpret_cast<char*>(buffer + startIndex), bytesRead -
00045     startIndex);
00046
00047     if (received.empty()) return;
00048
00049     // Debug: Print raw hex values
00050     std::stringstream hexDump;
00051     hexDump << "Raw bytes: ";
00052     for (int i = 0; i < bytesRead; i++) {
00053         hexDump << std::hex << std::setfill('0') << std::setw(2)
00054             << static_cast<int>(buffer[i]) << " ";
00055     }
00056     uart_print(hexDump.str(), VerbosityLevel::DEBUG);
00057
00058     // Find frame boundaries
00059     size_t headerPos = received.find(FRAME_BEGIN);
00060     size_t footerPos = received.find(FRAME_END);
00061
00062     if (headerPos != std::string::npos && footerPos != std::string::npos && footerPos > headerPos) {
00063         std::string frameData = received.substr(headerPos, footerPos + FRAME_END.length() -
00064         headerPos);
00065         uart_print("Extracted frame (length=" + std::to_string(frameData.length()) + "): " +
00066         frameData, VerbosityLevel::DEBUG);
00067         frame_process(frameData, Interface::LORA);
00068     } else {
00069         uart_print("No valid frame found in received data", VerbosityLevel::WARNING);
00070     }
00071 }
00072
00073 void handle_uart_input() {
00074     static std::string uartBuffer;
00075
00076     while (uart_is_readable(DEBUG_UART_PORT)) {
00077         char c = uart_getc(DEBUG_UART_PORT);
00078
00079         if (c == '\r' || c == '\n') {
00080             uart_print("Received UART string: " + uartBuffer, VerbosityLevel::DEBUG);
00081             frame_process(uartBuffer, Interface::UART);
00082             uartBuffer.clear();
00083         } else {
00084             uartBuffer += c;
00085         }
00086     }
00087 }
00088
00089 }
```

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.

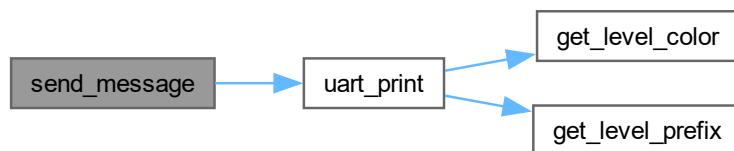
Parameters

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

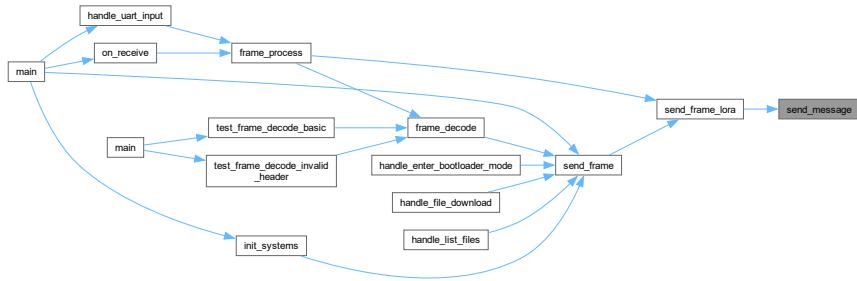
Converts the outgoing string to a C-style string, adds destination and local addresses, and sends the message using LoRa. Prints a log message to the UART.

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

Here is the call graph for this function:



Here is the caller graph for this function:

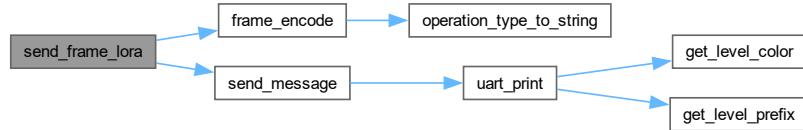


8.47.2.2 send_frame_lora()

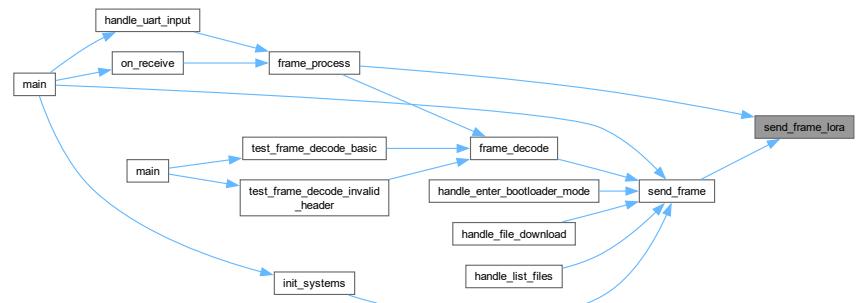
```
void send_frame_lora (
    const Frame & frame)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

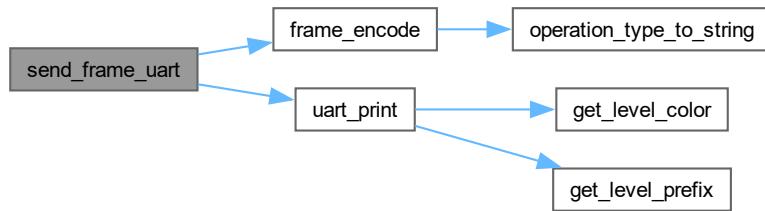


8.47.2.3 send_frame_uart()

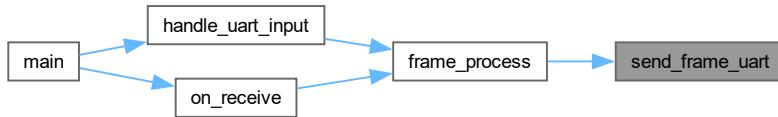
```
void send_frame_uart (
    const Frame & frame)
```

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

Here is the call graph for this function:



Here is the caller graph for this function:

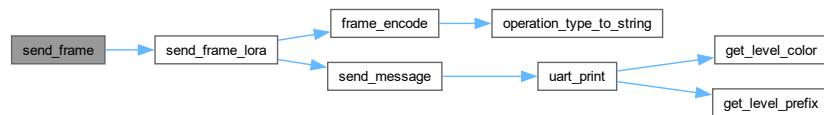


8.47.2.4 send_frame()

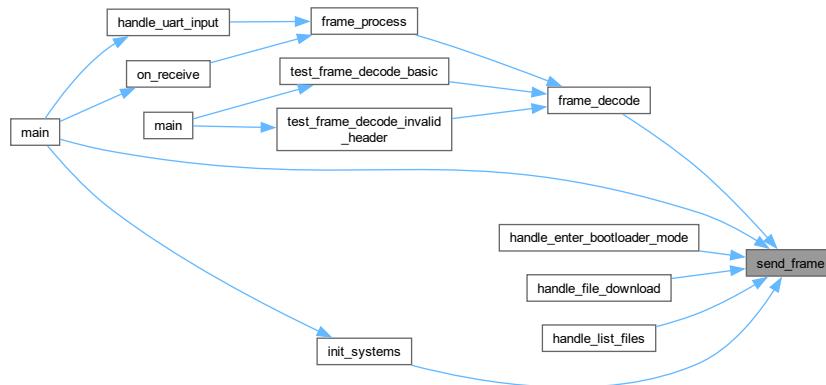
```
void send_frame (
    const Frame & frame)
```

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

<i>data</i>	The data to send.
<i>length</i>	The length of the data.

Splits the data into chunks of MAX_PKT_SIZE and sends each chunk as a separate LoRa packet.

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

8.48 send.cpp

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

```
00001 #include "communication.h"
00002
00003
00008
00015 void send_message(string outgoing)
00016 {
00017     int n = outgoing.length();
00018     char send[n + 1];
00019     strcpy(send, outgoing.c_str());
00020
00021     LoRa.beginPacket();      // start packet
00022     LoRa.write(lora_address_remote); // add destination address
00023     LoRa.write(lora_address_local); // add sender address
00024     LoRa.print(send);        // add payload
00025     LoRa.endPacket(false);   // finish packet and send it
00026
00027     std::string messageToLog = "Sent message of size " + std::to_string(n);
00028     messageToLog += " to 0x" + std::to_string(lora_address_remote);
```

```

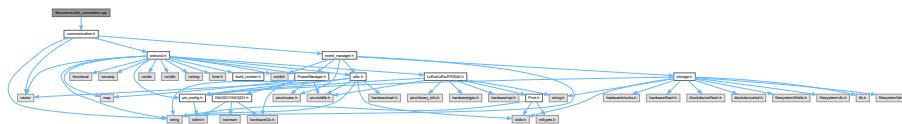
00029     messageToLog += " containing: " + string(send);
00030
00031     uart_print(messageToLog, VerbosityLevel::DEBUG);
00032
00033     LoRa.flush();
00034 }
00035
00036
00037 void send_frame_lora(const Frame& frame) {
00038     std::string encodedFrame = frame_encode(frame);
00039     send_message(encodedFrame);
00040 }
00041
00042 void send_frame_uart(const Frame& frame) {
00043     std::string encodedFrame = frame_encode(frame);
00044     uart_print(encodedFrame);
00045 }
00046
00047 [[deprecated("Use send_frame_lora or send_frame_uart instead")]]
00048 void send_frame(const Frame& frame) {
00049     send_frame_lora(frame);
00050 }
00051
00052
00053 void split_and_send_message(const uint8_t* data, size_t length)
00054 {
00055     const size_t MAX_PKT_SIZE = 255;
00056     size_t offset = 0;
00057     while (offset < length)
00058     {
00059         size_t chunkSize = ((length - offset) < MAX_PKT_SIZE) ? (length - offset) : MAX_PKT_SIZE;
00060         LoRa.beginPacket();
00061         LoRa.write(data[offset], chunkSize);
00062         LoRa.endPacket();
00063         offset += chunkSize;
00064         sleep_ms(100);
00065     }
00066 }
00067
00068

```

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

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

Returns

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

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

Here is the caller graph for this function:



8.49.2.2 value_unit_type_to_string()

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

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

Parameters

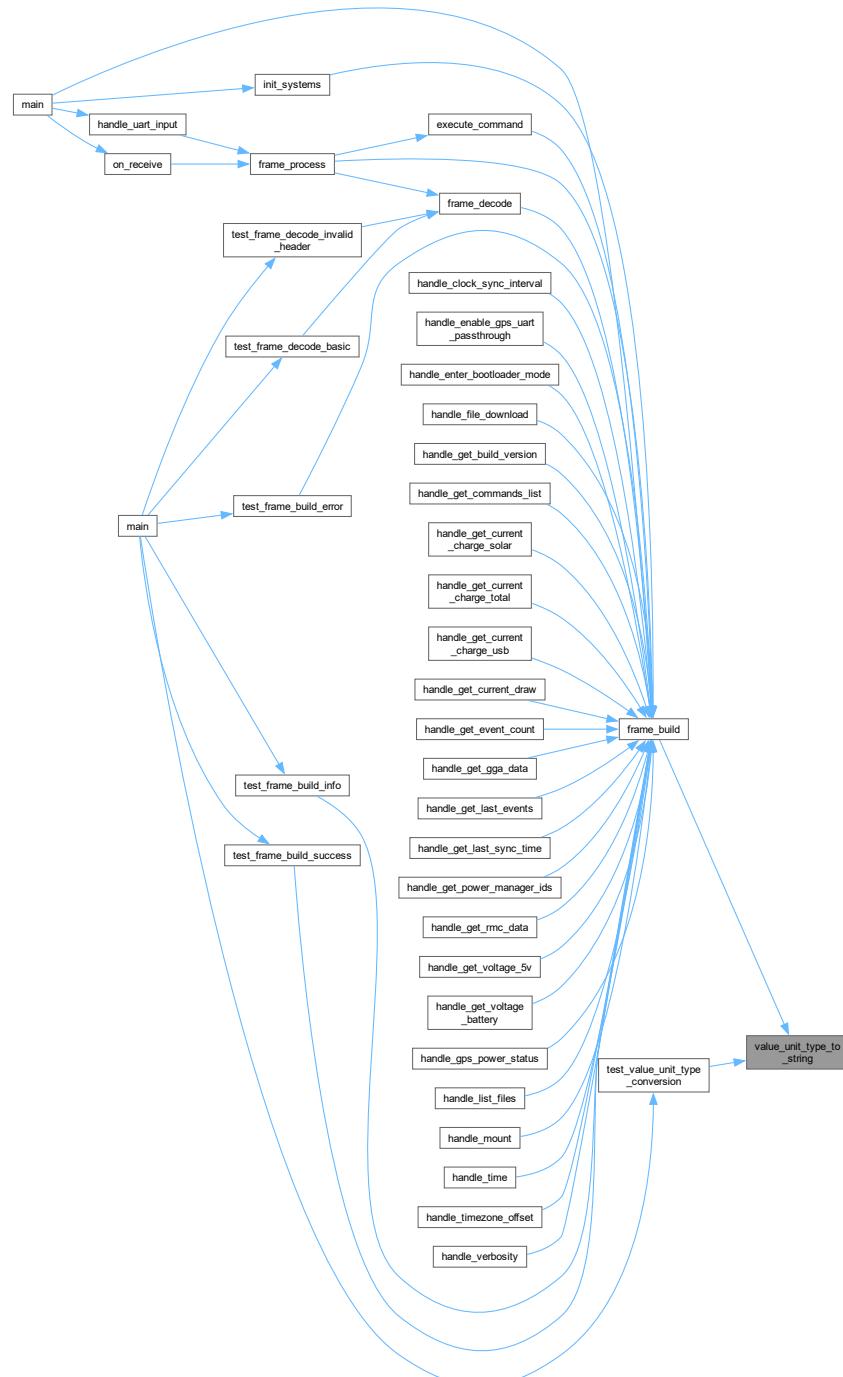
<i>unit</i>	The ValueUnit to convert.
-------------	---

Returns

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

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

Here is the caller graph for this function:



8.49.2.3 operation_type_to_string()

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

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

Parameters

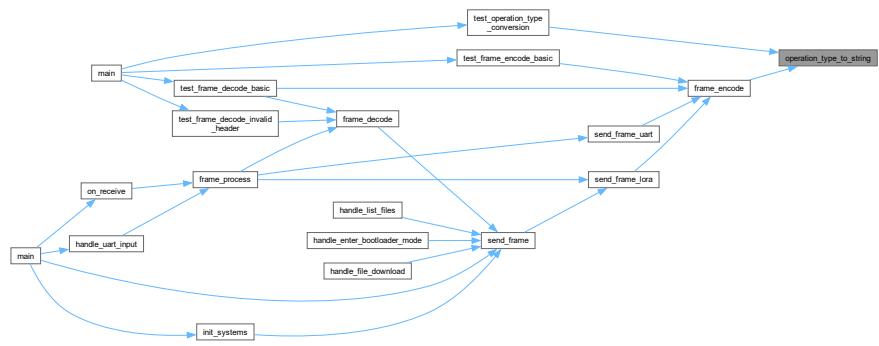
<code>type</code>	The OperationType to convert.
-------------------	---

Returns

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

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

Here is the caller graph for this function:



8.49.2.4 string_to_operation_type()

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

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

Parameters

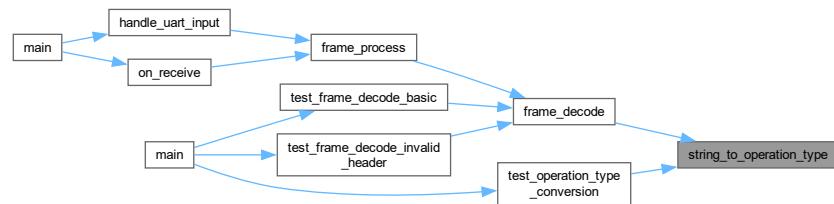
<code>str</code>	The string to convert.
------------------	------------------------

Returns

The `OperationType` corresponding to the string. Defaults to GET if the string is not recognized.

Definition at line 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

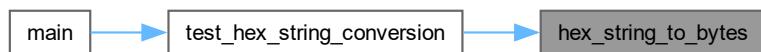
<i>hexString</i>	The hex string to convert.
------------------	----------------------------

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.](#)

```

00001 #include "communication.h"
00002
00003
00008
00014 std::string exception_type_to_string(ExceptionType type) {
00015     switch (type) {
00016         case ExceptionType::NOT_ALLOWED:      return "NOT ALLOWED";
00017         case ExceptionType::INVALID_PARAM:    return "INVALID PARAM";
00018         case ExceptionType::INVALID_OPERATION: return "INVALID OPERATION";
00019         case ExceptionType::PARAM_UNNECESSARY: return "PARAM UNECESSARY";
00020         case ExceptionType::NONE:             return "NONE";
00021         default:                           return "UNKNOWN EXCEPTION";
00022     }
00023 }
00024
00025
00031 std::string value_unit_type_to_string(ValueUnit unit) {
00032     switch (unit) {
00033         case ValueUnit::UNDEFINED:   return "";
00034         case ValueUnit::SECOND:      return "s";
00035         case ValueUnit::VOLT:        return "V";
00036         case ValueUnit::BOOL:        return "";
00037         case ValueUnit::DATETIME:   return "";
00038         case ValueUnit::TEXT:        return "";
00039         case ValueUnit::MILIAMP:    return "mA";
00040         default:                   return "";
00041     }
00042 }
00043
00044
00050 std::string operation_type_to_string(OperationType type) {
00051     switch (type) {
00052         case OperationType::GET:   return "GET";
00053         case OperationType::SET:   return "SET";
00054         case OperationType::ANS:   return "ANS";
00055         case OperationType::ERR:   return "ERR";
00056         case OperationType::INF:   return "INF";
00057         default:                  return "UNKNOWN";
00058     }
00059 }
00060
00061
00067 OperationType string_to_operation_type(const std::string& str) {
00068     if (str == "GET") return OperationType::GET;
00069     if (str == "SET") return OperationType::SET;
00070     if (str == "ANS") return OperationType::ANS;
00071     if (str == "ERR") return OperationType::ERR;
00072     if (str == "INF") return OperationType::INF;
00073     return OperationType::GET; // Default to GET
00074 }
00075
00081 std::vector<uint8_t> hex_string_to_bytes(const std::string& hexString) {
00082     std::vector<uint8_t> bytes;
00083     for (size_t i = 0; i < hexString.length(); i += 2) {
00084         std::string byteString = hexString.substr(i, 2);
00085         unsigned int byte;
00086         std::stringstream ss;
00087         ss << std::hex << byteString;
00088         ss >> byte;
00089         bytes.push_back(static_cast<uint8_t>(byte));
00090     }
00091     return bytes;
00092 }
```

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 Kabisat firmware.

This file contains the implementation for logging events, managing event storage, and checking for specific events such as power status changes.

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

8.51.2 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.

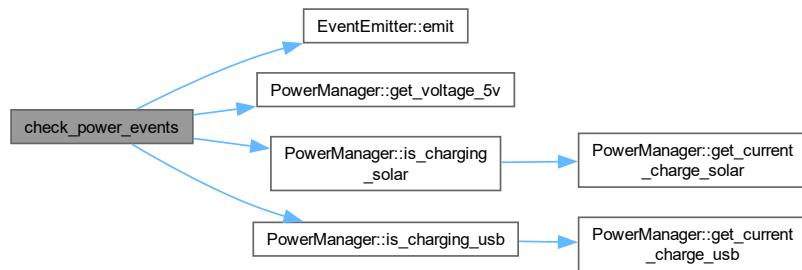
Parameters

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

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

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.](#)

```
00001 #include "event_manager.h"
00002 #include <cstdio>
00003 #include "protocol.h"
00004 #include "pico/multicore.h"
00005 #include "communication.h"
00006 #include "utils.h"
00007 #include "DS3231.h"
00008
00015
00016
00020 volatile uint16_t eventLogId = 0;
00021
00025 static PowerEvent lastPowerState = PowerEvent::LOW_BATTERY;
00026
00030 static constexpr float FALL_RATE_THRESHOLD = -0.02f;
00031
00035 static constexpr int FALLING_TREND_REQUIRED = 3;
00036
00040 static constexpr float VOLTAGE_LOW_THRESHOLD = 4.7f;
00041
00045 static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f;
00046
```

```

00050 static int fallingTrendCount = 0;
00051
00055 bool lastSolarState = false;
00056
00060 bool lastUSBState = false;
00061
00065 extern DS3231 systemClock;
00066
00070 EventManagerImpl eventManager;
00071
00072
00080 void EventManager::log_event(uint8_t group, uint8_t event) {
00081     mutex_enter_blocking(&eventMutex);
00082
00083     EventLog& log = events[writeIndex];
00084     log.id = nextEventId++;
00085     log.timestamp = systemClock.get_unix_time();
00086     log.group = group;
00087     log.event = event;
00088
00089     // Print event immediately
00090     uart_print(log.to_string(), VerbosityLevel::EVENT);
00091
00092     writeIndex = (writeIndex + 1) % EVENT_BUFFER_SIZE;
00093     if (eventCount < EVENT_BUFFER_SIZE) {
00094         eventCount++;
00095     }
00096
00097     // Set persistence flag on buffer full or power events
00098     if (eventCount == EVENT_BUFFER_SIZE || 
00099         (group == static_cast<uint8_t>(EventGroup::POWER) &&
00100         event == static_cast<uint8_t>(PowerEvent::POWER_FALLING))) {
00101         needsPersistence = true;
00102         save_to_storage();
00103     }
00104
00105     mutex_exit(&eventMutex);
00106 }
00107
00108
00114 const EventLog& EventManager::get_event(size_t index) const {
00115     static const EventLog emptyEvent = {0, 0, 0, 0}; // Initialize {id, timestamp, group, event}
00116     if (index >= eventCount) {
00117         return emptyEvent;
00118     }
00119
00120     // Calculate actual index in circular buffer
00121     size_t actualIndex;
00122     if (eventCount == EVENT_BUFFER_SIZE) {
00123         actualIndex = (writeIndex + index) % EVENT_BUFFER_SIZE;
00124     } else {
00125         actualIndex = index;
00126     }
00127
00128     return events[actualIndex];
00129 }
00130
00131
00139 void check_power_events(PowerManager& pm) {
00140     float currentVoltage = pm.get_voltage_5v();
00141     static float previousVoltage = 0.0f;
00142     float delta = currentVoltage - previousVoltage;
00143     previousVoltage = currentVoltage;
00144
00145     if (delta < FALL_RATE_THRESHOLD) {
00146         fallingTrendCount++;
00147     } else {
00148         fallingTrendCount = 0;
00149     }
00150
00151     if (fallingTrendCount >= FALLING_TREND_REQUIRED) {
00152         lastPowerState = PowerEvent::POWER_FALLING;
00153         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_FALLING);
00154     }
00155
00156     if (currentVoltage < PowerManager::VOLTAGE_LOW_THRESHOLD &&
00157         lastPowerState != PowerEvent::LOW_BATTERY) {
00158         lastPowerState = PowerEvent::LOW_BATTERY;
00159         EventEmitter::emit(EventGroup::POWER, PowerEvent::LOW_BATTERY);
00160     }
00161     else if (currentVoltage > PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&
00162             lastPowerState != PowerEvent::OVERCHARGE) {
00163         lastPowerState = PowerEvent::OVERCHARGE;
00164         EventEmitter::emit(EventGroup::POWER, PowerEvent::OVERCHARGE);
00165     }
00166     else if (currentVoltage >= PowerManager::VOLTAGE_LOW_THRESHOLD &&
00167             currentVoltage <= PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&

```

```

00168     lastPowerState != PowerEvent::POWER_NORMAL) {
00169         lastPowerState = PowerEvent::POWER_NORMAL;
00170         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_NORMAL);
00171     }
00172
00173     // Check solar charging state
00174     bool currentSolarState = pm.is_charging_solar();
00175     if (currentSolarState != lastSolarState) {
00176         if (currentSolarState) {
00177             EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_ACTIVE);
00178         } else {
00179             EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_INACTIVE);
00180         }
00181         lastSolarState = currentSolarState;
00182     }
00183
00184     // Check USB connection state
00185     bool currentUSBState = pm.is_charging_usb();
00186     if (currentUSBState != lastUSBState) {
00187         if (currentUSBState) {
00188             EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_CONNECTED);
00189         } else {
00190             EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_DISCONNECTED);
00191         }
00192         lastUSBState = currentUSBState;
00193     }
00194 }
```

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.txt"

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.txt"
```

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.

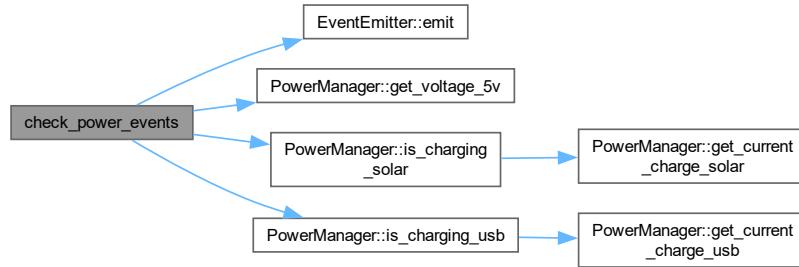
Parameters

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

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

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.](#)

```
00001 #ifndef EVENT_MANAGER_H
00002 #define EVENT_MANAGER_H
00003
00004 #include "PowerManager.h"
00005 #include <cstdint>
00006 #include <string>
00007 #include "pico/mutex.h"
00008 #include "storage.h"
00009 #include "utils.h"
00010
00011 #define EVENT_BUFFER_SIZE 10
00012 #define EVENT_LOG_FILE "/event_log.txt"
00013
00014 // Event Groups
00015 enum class EventGroup : uint8_t {
00016     SYSTEM = 0x00,
00017     POWER = 0x01,
00018     COMMS = 0x02,
00019     GPS = 0x03,
00020     CLOCK = 0x04
00021 };
00022
00023 // System Events
00024 enum class SystemEvent : uint8_t {
00025     BOOT = 0x01,
00026     SHUTDOWN = 0x02,
00027     WATCHDOG_RESET = 0x03,
00028     CORE1_START = 0x04,
00029     CORE1_STOP = 0x05
00030 };
00031
00032 // Power Events
00033 enum class PowerEvent : uint8_t {
00034     LOW_BATTERY = 0x01,
00035     OVERCHARGE = 0x02,
00036     POWER_FALLING = 0x03,
00037     POWER_NORMAL = 0x04,
00038     SOLAR_ACTIVE = 0x05,
00039     SOLAR_INACTIVE = 0x06,
00040     USB_CONNECTED = 0x07,
```

```

00041     USB_DISCONNECTED = 0x08
00042 };
00043
00044 // Communication Events
00045 enum class CommsEvent : uint8_t {
00046     RADIO_INIT      = 0x01,
00047     RADIO_ERROR     = 0x02,
00048     MSG_RECEIVED    = 0x03,
00049     MSG_SENT        = 0x04,
00050     UART_ERROR      = 0x06
00051 };
00052
00053 // GPS Events
00054 enum class GPSEvent : uint8_t {
00055     LOCK           = 0x01,
00056     LOST            = 0x02,
00057     ERROR           = 0x03,
00058     POWER_ON        = 0x04,
00059     POWER_OFF       = 0x05,
00060     DATA_READY      = 0x06,
00061     PASS_THROUGH_START = 0x07,
00062     PASS_THROUGH_END  = 0x08,
00063 };
00064
00065 // Clock Events
00066 enum class ClockEvent : uint8_t {
00067     CHANGED         = 0x01,
00068     GPS_SYNC        = 0x02
00069 };
00070
00071
00072 class EventLog {
00073 public:
00074     uint16_t id;
00075     uint32_t timestamp;
00076     uint8_t group;
00077     uint8_t event;
00078
00079     std::string to_string() const {
00080         char buffer[256];
00081         snprintf(buffer, sizeof(buffer),
00082                 "EventLog: id=%u, timestamp=%lu, group=%u, event=%u",
00083                 id, timestamp, group, event);
00084         return std::string(buffer);
00085     }
00086 } __attribute__((packed));
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101 class EventManager {
00102 public:
00103     EventManager()
00104         : eventCount(0)
00105         , writeIndex(0)
00106         , nextEventId(0)
00107         , needsPersistence(false)
00108     {
00109         mutex_init(&eventMutex);
00110     }
00111
00112     virtual ~EventManager() = default;
00113
00114     virtual void init() {
00115         load_from_storage();
00116     }
00117
00118     void log_event(uint8_t group, uint8_t event);
00119
00120     const EventLog& get_event(size_t index) const;
00121
00122     size_t get_event_count() const { return eventCount; }
00123
00124     virtual bool save_to_storage() = 0;
00125
00126     virtual bool load_from_storage() = 0;
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161 protected:
00162     EventLog events[EVENT_BUFFER_SIZE];
00163     size_t eventCount;
00164     size_t writeIndex;
00165     mutex_t eventMutex;
00166     volatile uint16_t nextEventId;
00167     bool needsPersistence;
00168 };
00169
00170
00171 class EventManagerImpl : public EventManager {
00172 public:

```

```

00181     EventManagerImpl() {
00182         init(); // Safe to call virtual functions here
00183     }
00184
00190     bool save_to_storage() override {
00191         if(!sd_card_mounted) {
00192             bool status = fs_init();
00193             if(!status) {
00194                 return false;
00195             }
00196         }
00197         FILE *file = fopen(EVENT_LOG_FILE, "a");
00198         if (file) {
00199             for (size_t i = 0; i < eventCount; i++) {
00200                 fwrite(&events[i], sizeof(EventLog), 1, file);
00201             }
00202             fclose(file);
00203             needsPersistence = false;
00204             uart_print("Events saved to storage", VerboseLevel::INFO);
00205             return true;
00206         }
00207     }
00208
00214     bool load_from_storage() override {
00215         // TODO: Implement based on chosen storage (SD/EEPROM)
00216         return false;
00217     }
00218 };
00219
00220
00224 extern EventManagerImpl eventManager;
00225
00230 class EventEmitter {
00231     public:
00238     template<typename T>
00239     static void emit(EventGroup group, T event) {
00240         eventManager.log_event(
00241             static_cast<uint8_t>(group),
00242             static_cast<uint8_t>(event)
00243         );
00244     }
00245 };
00246
00247
00252 void check_power_events(PowerManager& pm);
00253
00254 #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.](#)

```
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/gps_collector.cpp
00002 #include "lib/location/gps_collector.h"
00003 #include "utils.h"
00004 #include "pico/time.h"
00005 #include "lib/location/NMEA/nmea_data.h"
00006 #include "event_manager.h"
00007 #include <vector>
00008 #include <ctime>
00009 #include <cstring>
00010 #include "DS3231.h"
00011 #include <sstream>
00012
00013 #define MAX_RAW_DATA_LENGTH 1024
00014
00015 extern NMEAData nmea_data;
```

```

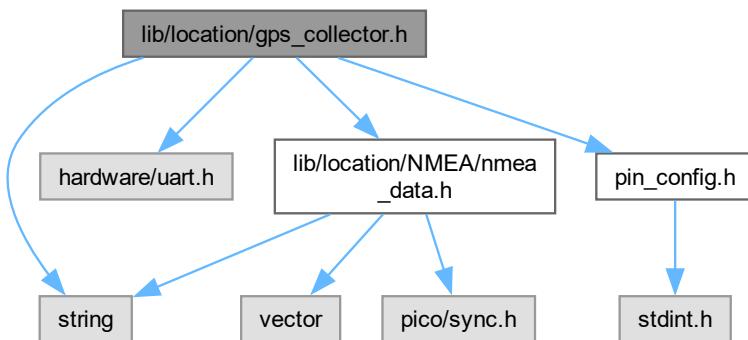
00016
00017 std::vector<std::string> splitString(const std::string& str, char delimiter) {
00018     std::vector<std::string> tokens;
00019     std::stringstream ss(str);
00020     std::string token;
00021     while (std::getline(ss, token, delimiter)) {
00022         tokens.push_back(token);
00023     }
00024     return tokens;
00025 }
00026
00027 void collect_gps_data() {
00028     static char raw_data_buffer[MAX_RAW_DATA_LENGTH];
00029     static int raw_data_index = 0;
00030
00031     while (uart_is_readable(GPS_UART_PORT)) {
00032         char c = uart_getc(GPS_UART_PORT);
00033
00034         if (c == '\r' || c == '\n') {
00035             // End of message
00036             if (raw_data_index > 0) {
00037                 raw_data_buffer[raw_data_index] = '\0';
00038                 std::string message(raw_data_buffer);
00039                 raw_data_index = 0;
00040
00041                 // Split the message into tokens
00042                 std::vector<std::string> tokens = splitString(message, ',');
00043
00044                 // Update the global vectors based on the sentence type
00045                 if (message.find("$GPRMC") == 0) {
00046                     nmea_data.update_rmc_tokens(tokens);
00047                 } else if (message.find("$GPGGA") == 0) {
00048                     nmea_data.update_gga_tokens(tokens);
00049                 }
00050             }
00051         } else {
00052             // Append to buffer
00053             if (raw_data_index < MAX_RAW_DATA_LENGTH - 1) {
00054                 raw_data_buffer[raw_data_index++] = c;
00055             } else {
00056                 raw_data_index = 0;
00057             }
00058         }
00059     }
00060 }

```

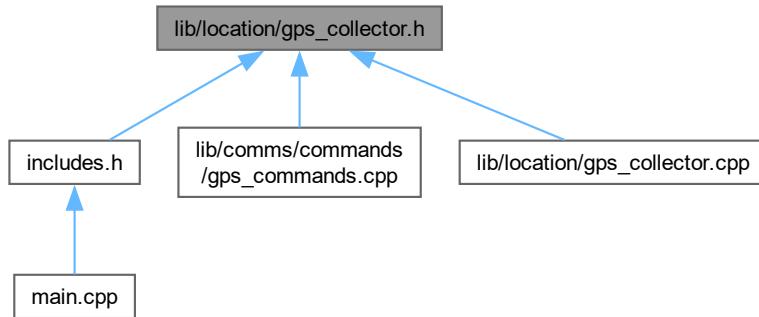
8.57 lib/location/gps_collector.h File Reference

```
#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:



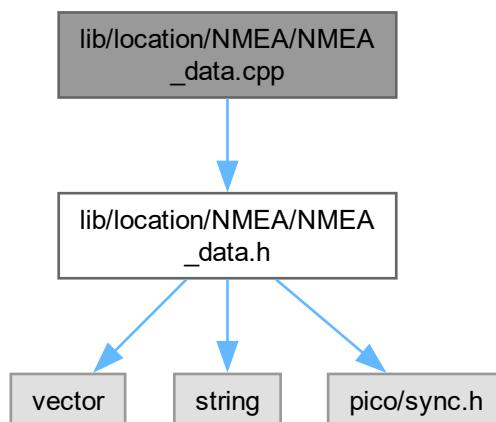
8.58 gps_collector.h

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

```
00001 #ifndef GPS_COLLECTOR_H
00002 #define GPS_COLLECTOR_H
00003
00004 #include <string>
00005 #include "hardware/uart.h"
00006 #include "lib/location/NMEA/nmea_data.h" // Include the new header
00007 #include "pin_config.h"
00008
00009 // Function to collect GPS data from the UART
00010 void collect_gps_data();
00011
00012 #endif
```

8.59 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](#).

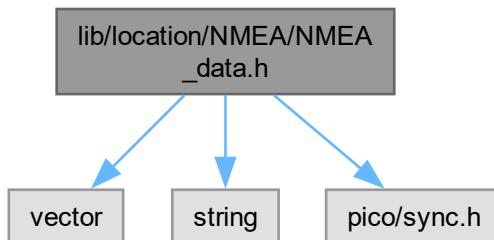
8.60 NMEA_data.cpp

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

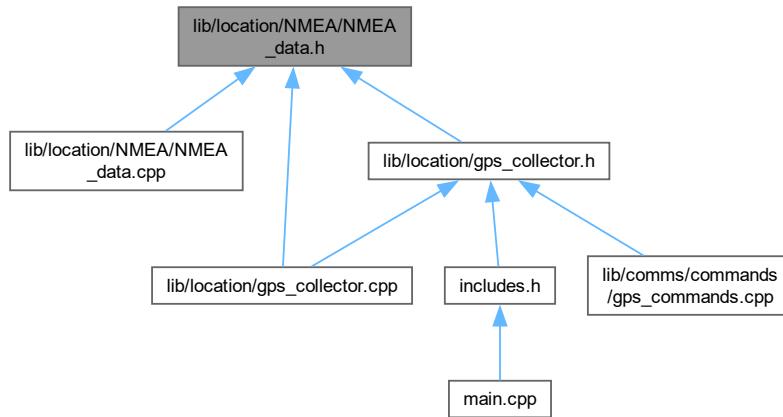
```
00001 #include "lib/location/NMEA/NMEA_data.h"
00002
00003 NMEAData nmea_data; // Define the global instance
00004
00005 NMEAData::NMEAData() {
00006     mutex_init(&rmc_mutex_);
00007     mutex_init(&gga_mutex_);
00008 }
00009
00010 void NMEAData::update_rmc_tokens(const std::vector<std::string>& tokens) {
00011     mutex_enter_blocking(&rmc_mutex_);
00012     rmc_tokens_ = tokens;
00013     mutex_exit(&rmc_mutex_);
00014 }
00015
00016 void NMEAData::update_gga_tokens(const std::vector<std::string>& tokens) {
00017     mutex_enter_blocking(&gga_mutex_);
00018     gga_tokens_ = tokens;
00019     mutex_exit(&gga_mutex_);
00020 }
00021
00022 std::vector<std::string> NMEAData::get_rmc_tokens() const {
00023     mutex_enter_blocking(const_cast<mutex_t*>(&rmc_mutex_));
00024     std::vector<std::string> copy = rmc_tokens_;
00025     mutex_exit(const_cast<mutex_t*>(&rmc_mutex_));
00026     return copy;
00027 }
00028
00029 std::vector<std::string> NMEAData::get_gga_tokens() const {
00030     mutex_enter_blocking(const_cast<mutex_t*>(&gga_mutex_));
00031     std::vector<std::string> copy = gga_tokens_;
00032     mutex_exit(const_cast<mutex_t*>(&gga_mutex_));
00033     return copy;
00034 }
```

8.61 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.](#)

```

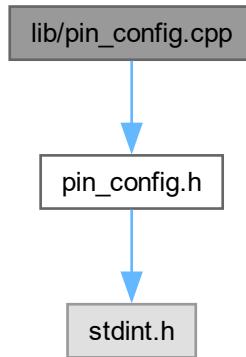
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/nmea_data.h
00002 #ifndef NMEA_DATA_H
00003 #define NMEA_DATA_H
00004
00005 #include <vector>
00006 #include <string>
00007 #include "pico/sync.h"
00008
00009 class NMEAData {
00010 public:
00011     NMEAData();
00012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

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

8.63 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](#).

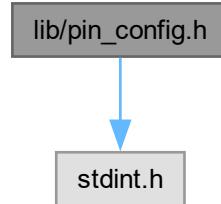
8.64 pin_config.cpp

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

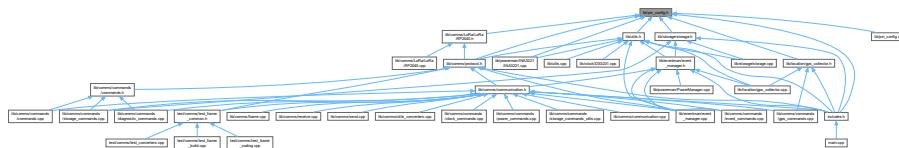
```
00001 #include "pin_config.h"
00002
00003 // LoRa constants
00004 const int lora_cs_pin = 17;           // LoRa radio chip select
00005 const int lora_reset_pin = 22;        // LoRa radio reset
00006 const int lora_irq_pin = 28;          // LoRa hardware interrupt pin
00007
00008 uint8_t lora_address_local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

8.65 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.](#)

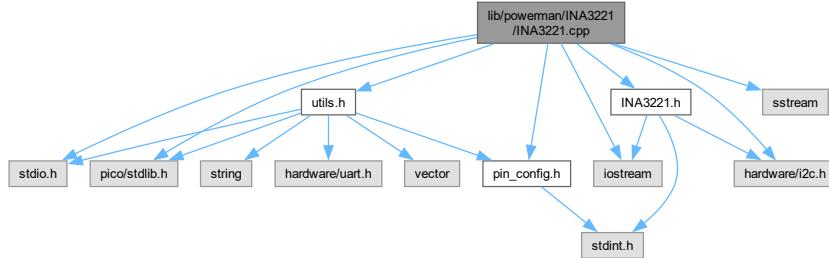
```
00001 // pin_config.h
00002 #include <stdint.h>
00003
00004 #ifndef PIN_CONFIG_H
00005 #define PIN_CONFIG_H
00006
00007 //DEBUG uart
00008 #define DEBUG_UART_PORT uart0
00009 #define DEBUG_UART_BAUD_RATE 115200
00010
00011 #define DEBUG_UART_TX_PIN 0
00012 #define DEBUG_UART_RX_PIN 1
00013
00014 #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 spil
00029 #define SD_MISO_PIN 12
00030 #define SD_MOSI_PIN 11
00031 #define SD_SCK_PIN 10
00032 #define SD_CS_PIN 13
00033 #define SD_CARD_DETECT_PIN 28
00034
00035 #define SX1278_MISO 16
00036 #define SX1278_CS 17
00037 #define SX1278_SCK 18
00038 #define SX1278_MOSI 19
00039
00040 #define SPI_PORT spi0
00041 #define READ_BIT 0x80
00042
00043 #define LORA_DEFAULT_SPI spi0
00044 #define LORA_DEFAULT_SPI_FREQUENCY 8E6
00045 #define LORA_DEFAULT_SS_PIN 17
00046 #define LORA_DEFAULT_RESET_PIN 22
00047 #define LORA_DEFAULT_DIO0_PIN 20
00048
00049 #define PA_OUTPUT_RFO_PIN 11
00050 #define PA_OUTPUT_PA_BOOST_PIN 12
00051
00052
00053
00054 // LoRa constants - declare as extern
00055 extern const int lora_cs_pin; // LoRa radio chip select
00056 extern const int lora_reset_pin; // LoRa radio reset
00057 extern const int lora_irq_pin; // LoRa hardware interrupt pin
00058 extern uint8_t lora_address_local; // address of this device
00059 extern uint8_t lora_address_remote; // destination to send to
00060
00061
00062 #endif // PIN_CONFIG_H
```

8.67 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(in3221_addr_t addr, i2c_inst_t* i2c)
00047     : _i2c_addr(addr), _i2c(i2c) {}
00048
00049
00056 bool INA3221::begin() {
00057     uart_print("INA3221 initializing...", VerboseLevel::DEBUG);

```

```
00058
00059     _shuntRes[0] = 10;
00060     _shuntRes[1] = 10;
00061     _shuntRes[2] = 10;
00062
00063     _filterRes[0] = 10;
00064     _filterRes[1] = 10;
00065     _filterRes[2] = 10;
00066
00067     uint16_t manuf_id = get_manufacturer_id();
00068     uint16_t die_id = get_die_id();
00069     std::stringstream ss;
00070     ss << "INA3221 Manufacturer ID: 0x" << std::hex << manuf_id
00071             << ", Die ID: 0x" << die_id << std::endl;
00072     uart_print(ss.str(), VerbosityLevel::INFO);
00073
00074     if (manuf_id == 0x5449 && die_id == 0x3220) {
00075         uart_print("INA3221 found and initialized.", VerbosityLevel::INFO);
00076         return true;
00077     } else {
00078         uart_print("INA3221 initialization failed. Incorrect IDs.", VerbosityLevel::ERROR);
00079         return false;
00080     }
00081
00082 }
00083
00084
00085 00090 void INA3221::reset(){
00086     conf_reg_t conf_reg;
00087
00088     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00089     conf_reg.reset = 1;
00090     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00091 }
00092
00093
00094
00095 00104 uint16_t INA3221::get_manufacturer_id() {
00096     uint16_t id = 0;
00097     _read(INA3221_REG_MANUF_ID, &id);
00098     return id;
00099 }
00100
00101
00102 00116 uint16_t INA3221::get_die_id() {
00103     uint16_t id = 0;
00104     _read(INA3221_REG_DIE_ID, &id);
00105     return id;
00106 }
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136 //configure
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178
00179
00180
```

```

00185 void INA3221::set_shunt_measurement_enable() {
00186     conf_reg_t conf_reg;
00187
00188     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00189     conf_reg.mode_shunt_en = 1;
00190     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00191 }
00192
00193
00198 void INA3221::set_shunt_measurement_disable() {
00199     conf_reg_t conf_reg;
00200
00201     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00202     conf_reg.mode_shunt_en = 0;
00203     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00204 }
00205
00206
00211 void INA3221::set_bus_measurement_enable() {
00212     conf_reg_t conf_reg;
00213
00214     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00215     conf_reg.mode_bus_en = 1;
00216     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00217 }
00218
00219
00224 void INA3221::set_bus_measurement_disable() {
00225     conf_reg_t conf_reg;
00226
00227     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00228     conf_reg.mode_bus_en = 0;
00229     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00230 }
00231
00232
00238 void INA3221::set_averaging_mode(ina3221_avg_mode_t mode) {
00239     conf_reg_t conf_reg;
00240
00241     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00242     conf_reg.avg_mode = mode;
00243     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00244 }
00245
00246
00252 void INA3221::set_bus_conversion_time(ina3221_conv_time_t convTime) {
00253     conf_reg_t conf_reg;
00254
00255     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00256     conf_reg.bus_conv_time = convTime;
00257     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00258 }
00259
00260
00266 void INA3221::set_shunt_conversion_time(ina3221_conv_time_t convTime) {
00267     conf_reg_t conf_reg;
00268
00269     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00270     conf_reg.shunt_conv_time = convTime;
00271     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00272 }
00273
00274
00275 //get measurement
00282 int32_t INA3221::get_shunt_voltage(ina3221_ch_t channel) {
00283     int32_t res;
00284     ina3221_reg_t reg;
00285     uint16_t val_raw = 0;
00286
00287     switch(channel){
00288         case INA3221_CH1:
00289             reg = INA3221_REG_CH1_SHUNTV;
00290             break;
00291         case INA3221_CH2:
00292             reg = INA3221_REG_CH2_SHUNTV;
00293             break;
00294         case INA3221_CH3:
00295             reg = INA3221_REG_CH3_SHUNTV;
00296             break;
00297     }
00298
00299     _read(reg, &val_raw);
00300
00301     res = (int16_t) (val_raw >> 3);
00302     res *= SHUNT_VOLTAGE_LSB_UV;
00303
00304     return res;

```

```
00305 }
00306
00307
00314 float INA3221::get_current_ma(ina3221_ch_t channel) {
00315     int32_t shunt_uV = 0;
00316     float current_A = 0;
00317
00318     shunt_uV = get_shunt_voltage(channel);
00319     current_A = shunt_uV / (int32_t)_shuntRes[channel] / 1000.0;;
00320     return current_A;
00321 }
00322
00323
00330 float INA3221::get_voltage(ina3221_ch_t channel) {
00331     float voltage_V = 0.0;
00332     ina3221_reg_t reg;
00333     uint16_t val_raw = 0;
00334
00335     switch(channel){
00336         case INA3221_CH1:
00337             reg = INA3221_REG_CH1_BUSV;
00338             break;
00339         case INA3221_CH2:
00340             reg = INA3221_REG_CH2_BUSV;
00341             break;
00342         case INA3221_CH3:
00343             reg = INA3221_REG_CH3_BUSV;
00344             break;
00345     }
00346
00347     _read(reg, &val_raw);
00348     voltage_V = val_raw / 1000.0;
00349     return voltage_V;
00350 }
00351
00352
00353 // alerts
00360 void INA3221::set_warn_alert_limit(ina3221_ch_t channel, float voltage_v) {
00361     ina3221_reg_t reg;
00362     uint16_t val = (uint16_t)(voltage_v * 1000); // Convert V to mV
00363
00364     switch(channel) {
00365         case INA3221_CH1:
00366             reg = INA3221_REG_CH1_WARNING_ALERT_LIM;
00367             break;
00368         case INA3221_CH2:
00369             reg = INA3221_REG_CH2_WARNING_ALERT_LIM;
00370             break;
00371         case INA3221_CH3:
00372             reg = INA3221_REG_CH3_WARNING_ALERT_LIM;
00373             break;
00374     }
00375     _write(reg, &val);
00376 }
00377
00378
00385 void INA3221::set_crit_alert_limit(ina3221_ch_t channel, float voltage_v) {
00386     ina3221_reg_t reg;
00387     uint16_t val = (uint16_t)(voltage_v * 1000); // Convert V to mV
00388
00389     switch(channel) {
00390         case INA3221_CH1:
00391             reg = INA3221_REG_CH1_CRIT_ALERT_LIM;
00392             break;
00393         case INA3221_CH2:
00394             reg = INA3221_REG_CH2_CRIT_ALERT_LIM;
00395             break;
00396         case INA3221_CH3:
00397             reg = INA3221_REG_CH3_CRIT_ALERT_LIM;
00398             break;
00399     }
00400     _write(reg, &val);
00401 }
00402
00403
00410 void INA3221::set_power_valid_limit(float voltage_upper_v, float voltage_lower_v) {
00411     uint16_t val;
00412
00413     val = (uint16_t)(voltage_upper_v * 1000);
00414     _write(INA3221_REG_PWR_VALID_HI_LIM, &val);
00415
00416     val = (uint16_t)(voltage_lower_v * 1000);
00417     _write(INA3221_REG_PWR_VALID_LO_LIM, &val);
00418 }
00419
00420
00426 void INA3221::enable_alerts() {
```

```

00427     masken_reg_t masken;
00428     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00429
00430     masken.warn_alert_ch1 = 1;
00431     masken.warn_alert_ch2 = 1;
00432     masken.warn_alert_ch3 = 1;
00433     masken.crit_alert_ch1 = 1;
00434     masken.crit_alert_ch2 = 1;
00435     masken.crit_alert_ch3 = 1;
00436     masken.pwr_valid_alert = 1;
00437
00438     _write(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00439 }
00440
00441
00442 bool INA3221::get_warn_alert(ina3221_ch_t channel) {
00443     masken_reg_t masken;
00444     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00445
00446     switch(channel) {
00447         case INA3221_CH1: return masken.warn_alert_ch1;
00448         case INA3221_CH2: return masken.warn_alert_ch2;
00449         case INA3221_CH3: return masken.warn_alert_ch3;
00450         default: return false;
00451     }
00452 }
00453
00454
00455 bool INA3221::get_crit_alert(ina3221_ch_t channel) {
00456     masken_reg_t masken;
00457     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00458
00459     switch(channel) {
00460         case INA3221_CH1: return masken.crit_alert_ch1;
00461         case INA3221_CH2: return masken.crit_alert_ch2;
00462         case INA3221_CH3: return masken.crit_alert_ch3;
00463         default: return false;
00464     }
00465 }
00466
00467
00468 bool INA3221::get_power_valid_alert() {
00469     masken_reg_t masken;
00470     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00471     return masken.pwr_valid_alert;
00472 }
00473
00474
00475 void INA3221::set_alert_latch(bool enable) {
00476     masken_reg_t masken;
00477     _read(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00478     masken.warn_alert_latch_en = enable;
00479     masken.crit_alert_latch_en = enable;
00480     _write(INA3221_REG_MASK_ENABLE, (uint16_t*)&masken);
00481 }
00482
00483
00484 // private
00485 void INA3221::_read(ina3221_reg_t reg, uint16_t *val) {
00486     uint8_t reg_buf = reg;
00487     uint8_t data[2];
00488
00489     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, &reg_buf, 1, true);
00490     if (ret != 1) {
00491         std::cerr << "Failed to write register address to I2C device." << std::endl;
00492         return;
00493     }
00494
00495     ret = i2c_read_blocking(MAIN_I2C_PORT, _i2c_addr, data, 2, false);
00496     if (ret != 2) {
00497         std::cerr << "Failed to read data from I2C device." << std::endl;
00498         return;
00499     }
00500
00501     *val = (data[0] << 8) | data[1];
00502 }
00503
00504
00505 void INA3221::_write(ina3221_reg_t reg, uint16_t *val) {
00506     uint8_t buf[3];
00507     buf[0] = reg;
00508     buf[1] = (*val >> 8) & 0xFF; // MSB
00509     buf[2] = (*val) & 0xFF; // LSB
00510
00511     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, buf, 3, false);
00512     if (ret != 3) {
00513         std::cerr << "Failed to write data to I2C device." << std::endl;
00514     }
00515 }
```

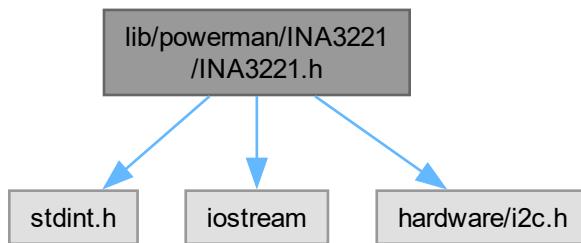
```
00548 }
00549 }
```

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

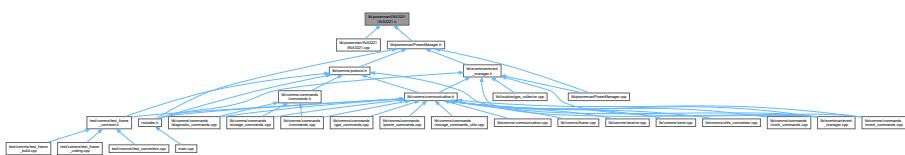
Header file for the [INA3221](#) triple-channel power monitor driver.

```
#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_ALEP` ,
`INA3221_REG_CH1_WARNING_ALERT_LIM` , `INA3221_REG_CH2_CRIT_ALERT_LIM` , `INA3221_REG_CH2_WARNING_A` ,
`INA3221_REG_CH3_CRIT_ALERT_LIM` , `INA3221_REG_SHUNTV_SUM` , `INA3221_REG_SHUNTV_SUM_LIM` ,
`INA3221_REG_MASK_ENABLE` ,
`INA3221_REG_PWR_VALID_HI_LIM` , `INA3221_REG_PWR_VALID_LO_LIM` , `INA3221_REG_MANUF_ID` = 0xFE , `INA3221_REG_DIE_ID` = 0xFF }

Register addresses for `INA3221`.
- enum `ina3221_conv_time_t`
 - `INA3221_REG_CONF_CT_140US` = 0 , `INA3221_REG_CONF_CT_204US` , `INA3221_REG_CONF_CT_332US` ,
`INA3221_REG_CONF_CT_588US` ,
`INA3221_REG_CONF_CT_1100US` , `INA3221_REG_CONF_CT_2116US` , `INA3221_REG_CONF_CT_4156US` ,
`INA3221_REG_CONF_CT_8244US` }

Conversion time settings.
- enum `ina3221_avg_mode_t`
 - `INA3221_REG_CONF_AVG_1` = 0 , `INA3221_REG_CONF_AVG_4` , `INA3221_REG_CONF_AVG_16` ,
`INA3221_REG_CONF_AVG_64` ,
`INA3221_REG_CONF_AVG_128` , `INA3221_REG_CONF_AVG_256` , `INA3221_REG_CONF_AVG_512` ,
`INA3221_REG_CONF_AVG_1024` }

Averaging mode settings.

Variables

- const int `INA3221_CH_NUM` = 3
Number of channels in `INA3221`.
- const int `SHUNT_VOLTAGE_LSB_UV` = 5
LSB value for shunt voltage measurements in microvolts.

8.69.1 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,
```

```

00088     INA3221_REG_CONF_AVG_1024
00089 } ina3221_avg_mode_t;
00090
00096 class INA3221 {
00097
00101     typedef struct {
00102         uint16_t mode_shunt_en:1;
00103         uint16_t mode_bus_en:1;
00104         uint16_t mode_continious_en:1;
00105         uint16_t shunt_conv_time:3;
00106         uint16_t bus_conv_time:3;
00107         uint16_t avg_mode:3;
00108         uint16_t ch3_en:1;
00109         uint16_t ch2_en:1;
00110         uint16_t ch1_en:1;
00111         uint16_t reset:1;
00112     } conf_reg_t;
00113
00117     typedef struct {
00118         uint16_t conv_ready:1;
00119         uint16_t timing_ctrl_alert:1;
00120         uint16_t pwr_valid_alert:1;
00121         uint16_t warn_alert_ch3:1;
00122         uint16_t warn_alert_ch2:1;
00123         uint16_t warn_alert_ch1:1;
00124         uint16_t shunt_sum_alert:1;
00125         uint16_t crit_alert_ch3:1;
00126         uint16_t crit_alert_ch2:1;
00127         uint16_t crit_alert_ch1:1;
00128         uint16_t crit_alert_latch_en:1;
00129         uint16_t warn_alert_latch_en:1;
00130         uint16_t shunt_sum_en_ch3:1;
00131         uint16_t shunt_sum_en_ch2:1;
00132         uint16_t shunt_sum_en_ch1:1;
00133         uint16_t reserved:1;
00134     } masken_reg_t;
00135
00136     i2c_inst_t* _i2c;
00137     // I2C address
00138     ina3221_addr_t _i2c_addr;
00139
00140     // Shunt resistance in mOhm
00141     uint32_t _shuntRes[INA3221_CH_NUM];
00142
00143     // Series filter resistance in Ohm
00144     uint32_t _filterRes[INA3221_CH_NUM];
00145
00146     // Value of Mask/Enable register.
00147     masken_reg_t _masken_reg;
00148
00149     // Reads 16 bytes from a register.
00150     void _read(ina3221_reg_t reg, uint16_t *val);
00151
00152     // Writes 16 bytes to a register.
00153     void _write(ina3221_reg_t reg, uint16_t *val);
00154
00155 public:
00156
00157     INA3221(ina3221_addr_t addr, i2c_inst_t* i2c);
00158     // Initializes INA3221
00159     bool begin();
00160
00161     // Gets a register value.
00162     uint16_t read_register(ina3221_reg_t reg);
00163
00164     // Resets INA3221
00165     void reset();
00166
00167     // Sets operating mode to power-down
00168     void set_mode_power_down();
00169
00170     // Sets operating mode to continious
00171     void set_mode_continious();
00172
00173     // Sets operating mode to triggered (single-shot)
00174     void set_mode_triggered();
00175
00176     // Enables shunt-voltage measurement
00177     void set_shunt_measurement_enable();
00178
00179     // Disables shunt-voltage mesurement
00180     void set_shunt_measurement_disable();
00181
00182     // Enables bus-voltage measurement
00183     void set_bus_measurement_enable();
00184
00185     // Disables bus-voltage measureement

```

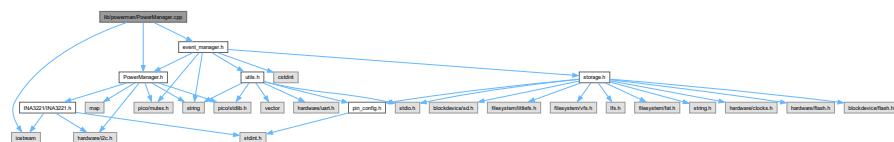
```

00186     void set_bus_measurement_disable();
00187
00188     // Sets averaging mode. Sets number of samples that are collected
00189     // and averaged together.
00190     void set_averaging_mode(ina3221_avg_mode_t mode);
00191
00192     // Sets bus-voltage conversion time.
00193     void set_bus_conversion_time(ina3221_conv_time_t convTime);
00194
00195     // Sets shunt-voltage conversion time.
00196     void set_shunt_conversion_time(ina3221_conv_time_t convTime);
00197
00198     // Gets manufacturer ID.
00199     // Should read 0x5449.
00200     uint16_t get_manufacturer_id();
00201
00202     // Gets die ID.
00203     // Should read 0x3220.
00204     uint16_t get_die_id();
00205
00206     // Gets shunt voltage in uV.
00207     int32_t get_shunt_voltage(ina3221_ch_t channel);
00208
00209     // Gets current in A.
00210     float get_current(ina3221_ch_t channel);
00211
00212     float get_current_ma(ina3221_ch_t channel);
00213
00214     // Gets bus voltage in V.
00215     float get_voltage(ina3221_ch_t channel);
00216
00217     void set_warn_alert_limit(ina3221_ch_t channel, float voltage_v);
00218     void set_crit_alert_limit(ina3221_ch_t channel, float voltage_v);
00219     void set_power_valid_limit(float voltage_upper_v, float voltage_lower_v);
00220     void enable_alerts();
00221     bool get_warn_alert(ina3221_ch_t channel);
00222     bool get_crit_alert(ina3221_ch_t channel);
00223     bool get_power_valid_alert();
00224     void set_alertLatch(bool enable);
00225 };
00226
00227 #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.

```

00001 #include "PowerManager.h"
00002 #include <iostream>
00003 #include "event_manager.h"
00004
00005 PowerManager::PowerManager(i2c_inst_t* i2c)
00006     : ina3221_(INA3221_ADDR40_GND, i2c) {
00007         recursive_mutex_init(&powerman_mutex_);
00008     };
00009

```

```

00010     bool PowerManager::initialize() {
00011         recursive_mutex_enter_blocking(&powerman_mutex_);
00012         initialized_ = ina3221_.begin();
00013
00014         if (initialized_) {
00015             // Set up alerts
00016             ina3221_.set_warn_alert_limit(INA3221_CH2, VOLTAGE_LOW_THRESHOLD);
00017             ina3221_.set_crit_alert_limit(INA3221_CH2, VOLTAGE_OVERCHARGE_THRESHOLD);
00018             ina3221_.set_power_valid_limit(VOLTAGE_OVERCHARGE_THRESHOLD, VOLTAGE_LOW_THRESHOLD);
00019             ina3221_.enable_alerts();
00020             ina3221_.set_alert_latch(true);
00021         }
00022
00023         recursive_mutex_exit(&powerman_mutex_);
00024         return initialized_;
00025     }
00026
00027     std::string PowerManager::read_device_ids() {
00028         if (!initialized_) return "noinit";
00029         recursive_mutex_enter_blocking(&powerman_mutex_);
00030         std::string MAN = "MAN " + std::to_string(ina3221_.get_manufacturer_id());
00031         std::string DIE = "DIE " + std::to_string(ina3221_.get_die_id());
00032         recursive_mutex_exit(&powerman_mutex_);
00033         return MAN + " - " + DIE;
00034     }
00035
00036     float PowerManager::get_voltage_battery() {
00037         if (!initialized_) return 0.0f;
00038         recursive_mutex_enter_blocking(&powerman_mutex_);
00039         float voltage = ina3221_.get_voltage(INA3221_CH1);
00040         recursive_mutex_exit(&powerman_mutex_);
00041         return voltage;
00042     }
00043
00044     float PowerManager::get_voltage_5v() {
00045         if (!initialized_) return 0.0f;
00046         recursive_mutex_enter_blocking(&powerman_mutex_);
00047         float voltage = ina3221_.get_voltage(INA3221_CH2);
00048         recursive_mutex_exit(&powerman_mutex_);
00049         return voltage;
00050     }
00051
00052     float PowerManager::get_current_charge_usb() {
00053         if (!initialized_) return 0.0f;
00054         recursive_mutex_enter_blocking(&powerman_mutex_);
00055         float current = ina3221_.get_current_ma(INA3221_CH1);
00056         recursive_mutex_exit(&powerman_mutex_);
00057         return current;
00058     }
00059
00060     float PowerManager::get_current_draw() {
00061         if (!initialized_) return 0.0f;
00062         recursive_mutex_enter_blocking(&powerman_mutex_);
00063         float current = ina3221_.get_current_ma(INA3221_CH2);
00064         recursive_mutex_exit(&powerman_mutex_);
00065         return current;
00066     }
00067
00068     float PowerManager::get_current_charge_solar() {
00069         if (!initialized_) return 0.0f;
00070         recursive_mutex_enter_blocking(&powerman_mutex_);
00071         float current = ina3221_.get_current_ma(INA3221_CH3);
00072         recursive_mutex_exit(&powerman_mutex_);
00073         return current;
00074     }
00075
00076     float PowerManager::get_current_charge_total() {
00077         if (!initialized_) return 0.0f;
00078         recursive_mutex_enter_blocking(&powerman_mutex_);
00079         float current = ina3221_.get_current_ma(INA3221_CH1) + ina3221_.get_current_ma(INA3221_CH3);
00080         recursive_mutex_exit(&powerman_mutex_);
00081         return current;
00082     }
00083
00084     void PowerManager::configure(const std::map<std::string, std::string>& config) {
00085         if (!initialized_) return;
00086         recursive_mutex_enter_blocking(&powerman_mutex_);
00087
00088         if (config.find("operating_mode") != config.end()) {
00089             if (config.at("operating_mode") == "continuous") {
00090                 ina3221_.set_mode_continuous();
00091             }
00092         }
00093
00094         if (config.find("averaging_mode") != config.end()) {
00095             int avg_mode = std::stoi(config.at("averaging_mode"));
00096             switch(avg_mode) {

```

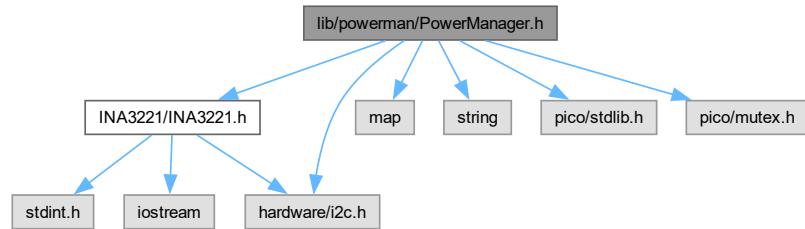
```

00097     case 1:
00098         ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_1);
00099         break;
00100     case 4:
00101         ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_4);
00102         break;
00103     case 16:
00104         ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_16);
00105         break;
00106     default:
00107         ina3221_.set_averaging_mode(INA3221_REG_CONF_AVG_16);
00108     }
00109 }
00110 recursive_mutex_exit(&powerman_mutex_);
00111 }
00112
00113 bool PowerManager::is_charging_solar() {
00114     if (!initialized_) return false;
00115     recursive_mutex_enter_blocking(&powerman_mutex_);
00116     bool active = get_current_charge_solar() > SOLAR_CURRENT_THRESHOLD;
00117     recursive_mutex_exit(&powerman_mutex_);
00118     return active;
00119 }
00120
00121 bool PowerManager::is_charging_usb() {
00122     if (!initialized_) return false;
00123     recursive_mutex_enter_blocking(&powerman_mutex_);
00124     bool connected = get_current_charge_usb() > USB_CURRENT_THRESHOLD;
00125     recursive_mutex_exit(&powerman_mutex_);
00126     return connected;
00127 }
00128
00129 bool PowerManager::check_power_alerts() {
00130     if (!initialized_) return false;
00131
00132     recursive_mutex_enter_blocking(&powerman_mutex_);
00133
00134     bool status_changed = false;
00135
00136     // Check warning alert (low battery)
00137     if (ina3221_.get_warn_alert(INA3221_CH2)) {
00138         EventEmitter::emit(EventGroup::POWER, PowerEvent::LOW_BATTERY);
00139         status_changed = true;
00140     }
00141
00142     // Check critical alert (overcharge)
00143     if (ina3221_.get_crit_alert(INA3221_CH2)) {
00144         EventEmitter::emit(EventGroup::POWER, PowerEvent::OVERCHARGE);
00145         status_changed = true;
00146     }
00147
00148     // Check power valid alert
00149     if (ina3221_.get_power_valid_alert()) {
00150         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_NORMAL);
00151         status_changed = true;
00152     }
00153
00154     recursive_mutex_exit(&powerman_mutex_);
00155     return status_changed;
00156 }
```

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.](#)

```

00001 #ifndef POWER_MANAGER_H
00002 #define POWER_MANAGER_H
00003
00004 #include "INA3221/INA3221.h"
00005 #include <map>
00006 #include <string>
00007 #include <hardware/i2c.h>
00008 #include "pico/stdlib.h"
00009 #include "pico/mutex.h"
00010
00011 class PowerManager {
00012 public:
00013     PowerManager(i2c_inst_t* i2c);
00014     bool initialize();
00015     std::string read_device_ids();
00016     float get_current_charge_solar();
00017     float get_current_charge_usb();
00018     float get_current_charge_total();
00019     float get_current_draw();
00020     float get_voltage_battery();
00021     float get_voltage_5v();
00022     void configure(const std::map<std::string, std::string>& config);
00023     bool is_charging_solar();
00024     bool is_charging_usb();
00025     bool check_power_alerts();
00026
00027     static constexpr float SOLAR_CURRENT_THRESHOLD = 50.0f; // mA
00028     static constexpr float USB_CURRENT_THRESHOLD = 50.0f; // mA
00029     static constexpr float VOLTAGE_LOW_THRESHOLD = 4.7f; // V
00030     static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f; // V
00031     static constexpr float FALL_RATE_THRESHOLD = -0.02f; // V/sample
00032     static constexpr int FALLING_TREND_REQUIRED = 3; // samples
00033
00034
  
```

```

00035 private:
00036     INA3221 ina3221_;
00037     bool initialized_;
00038     recursive_mutex_t powerman_mutex_;
00039     bool charging_solar_active_ = false;
00040     bool charging_usb_active_ = false;
00041 };
00042
00043 #endif // POWER_MANAGER_H

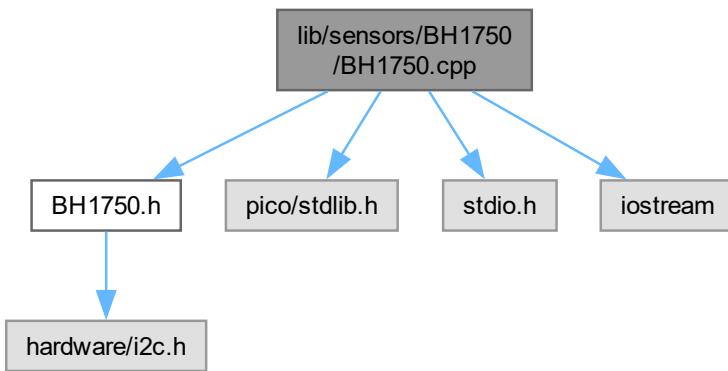
```

8.75 lib/sensors/BH1750/BH1750.cpp File Reference

```

#include "BH1750.h"
#include "pico/stl.h"
#include <stdio.h>
#include <iostream>
Include dependency graph for BH1750.cpp:

```



8.76 BH1750.cpp

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

```

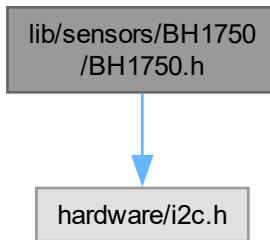
00001 #include "BH1750.h"
00002 #include "pico/stl.h"
00003 #include <stdio.h>
00004 #include <iostream>
00005
00006 BH1750::BH1750(uint8_t addr) : _i2c_addr(addr) {}
00007
00008 bool BH1750::begin(Mode mode) {
00009     write8(static_cast<uint8_t>(Mode::POWER_ON));
00010     write8(static_cast<uint8_t>(Mode::RESET));
00011     configure(mode);
00012     configure(BH1750::Mode::POWER_ON);
00013     uint8_t check = 0;
00014     uint8_t cmd = 0x10; // Continuously H-Resolution Mode
00015     if (_i2c_write_blocking(i2c1, _i2c_addr, &cmd, 1, false) == 1) {
00016         std::cout << "BH1750 sensor found at 0x" << std::hex << (int)_i2c_addr << std::endl;
00017         return true;
00018     }
00019     return false;
00020 }
00021
00022 void BH1750::configure(Mode mode) {

```

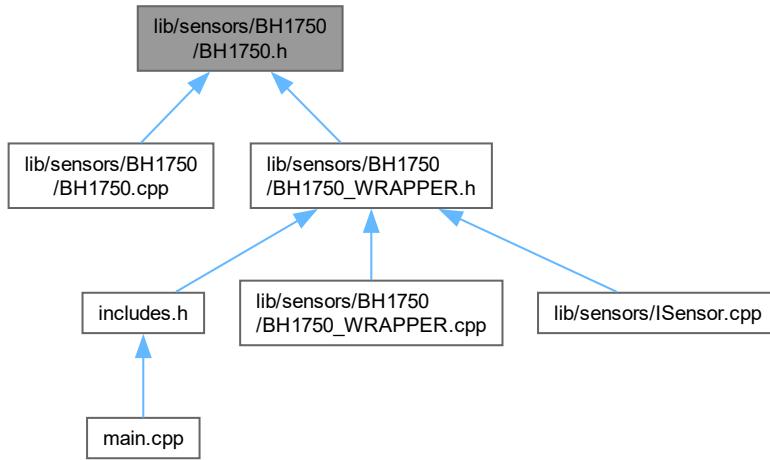
```
00023     uint8_t modeVal = static_cast<uint8_t>(mode);
00024     switch (mode) {
00025         case Mode::CONTINUOUS_HIGH_RES_MODE:
00026         case Mode::CONTINUOUS_HIGH_RES_MODE_2:
00027         case Mode::CONTINUOUS_LOW_RES_MODE:
00028         case Mode::ONE_TIME_HIGH_RES_MODE:
00029         case Mode::ONE_TIME_HIGH_RES_MODE_2:
00030         case Mode::ONE_TIME_LOW_RES_MODE:
00031             write8(modeVal);
00032             sleep_ms(10);
00033             break;
00034     default:
00035         printf("Invalid measurement mode\n");
00036         break;
00037     }
00038 }
00039
00040 float BH1750::get_light_level() {
00041     uint8_t buffer[2];
00042     i2c_read_blocking(i2c_default, _i2c_addr, buffer, 2, false);
00043     uint16_t level = (buffer[0] << 8) | buffer[1];
00044
00045     float lux = static_cast<float>(level) / 1.2f;
00046     return lux;
00047 }
00048
00049 void BH1750::write8(uint8_t data) {
00050     uint8_t buf[1] = {data};
00051     i2c_write_blocking(i2c_default, _i2c_addr, buf, 1, false);
00052 }
```

8.77 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

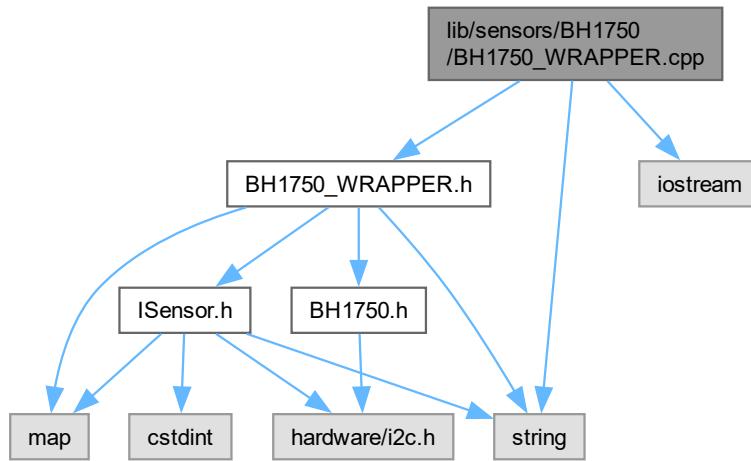
[Go to the documentation of this file.](#)

```
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.](#)

```

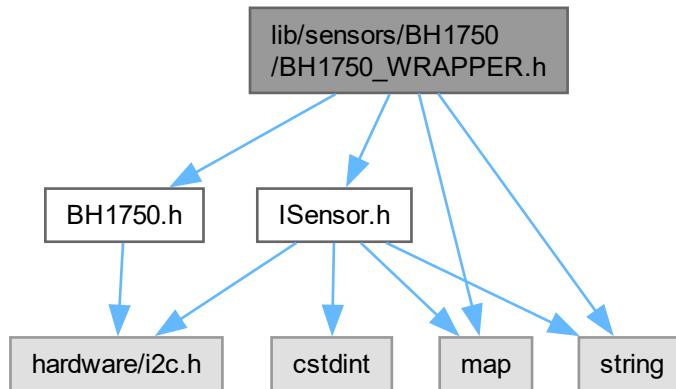
00001 // BH1750Wrapper.cpp
00002 #include "BH1750_WRAPPER.h"
00003 #include <string>
00004 #include <iostream>
00005
00006 BH1750Wrapper::BH1750Wrapper() {
00007     sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE);
00008 }
00009
00010 bool BH1750Wrapper::init() {
00011     initialized_ = sensor_.begin();
00012     return initialized_;
00013 }
00014
00015 float BH1750Wrapper::read_data(SensorDataTypeIdentifier type) {
00016     if (type == SensorDataTypeIdentifier::LIGHT_LEVEL) {
00017         return sensor_.get_light_level();
00018     }
00019     return 0.0f;
00020 }
00021
00022 bool BH1750Wrapper::is_initialized() const {
00023     return initialized_;
00024 }
00025
00026 SensorType BH1750Wrapper::get_type() const {
00027     return SensorType::LIGHT;
00028 }
00029
00030 bool BH1750Wrapper::configure(const std::map<std::string, std::string>& config) {
00031     for (const auto& [key, value] : config) {
00032         if (key == "measurement_mode") {
00033             if (value == "continuously_high_resolution") {
00034                 sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE);
00035             }
00036             else if (value == "continuously_high_resolution_2") {
00037                 sensor_.configure(BH1750::Mode::CONTINUOUS_HIGH_RES_MODE_2);
00038             }
00039             else if (value == "continuously_low_resolution") {
00040                 sensor_.configure(BH1750::Mode::CONTINUOUS_LOW_RES_MODE);
00041             }
00042         }
00043     }
00044 }
```

```

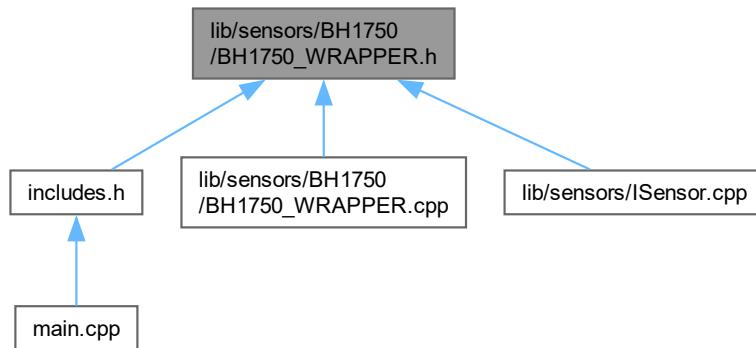
00042     else if (value == "one_time_high_resolution") {
00043         sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE);
00044     }
00045     else if (value == "one_time_high_resolution_2") {
00046         sensor_.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE_2);
00047     }
00048     else if (value == "one_time_low_resolution") {
00049         sensor_.configure(BH1750::Mode::ONE_TIME_LOW_RES_MODE);
00050     }
00051     else {
00052         std::cerr << "[BH1750Wrapper] Unknown measurement_mode value: " << value << std::endl;
00053         return false;
00054     }
00055 }
00056 // Handle additional configuration keys here
00057 else {
00058     std::cerr << "[BH1750Wrapper] Unknown configuration key: " << key << std::endl;
00059     return false;
00060 }
00061 }
00062 return true;
00063 }
```

8.81 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.](#)

```

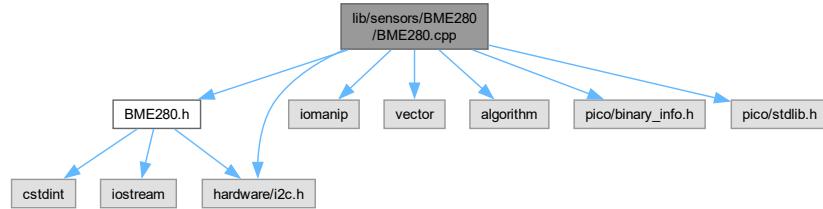
00001 #ifndef BH1750_WRAPPER_H
00002 #define BH1750_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "BH1750.h"
00006 #include <map>
00007 #include <string>
00008
00009 class BH1750Wrapper : public ISensor {
00010 private:
00011     BH1750 sensor_;
00012     bool initialized_ = false;
00013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

```

00054     initialized_ = true;
00055     std::cout << "BME280 sensor initialized_ successfully.\n";
00056     return true;
00057 }
00058
00059 void BME280::reset() {
00060     uint8_t buf[2] = { REG_RESET, 0xB6 };
00061     int ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00062     if (ret != 2) {
00063         std::cerr << "Failed to reset BME280 sensor.\n";
00064     }
00065     sleep_ms(10); // Wait for reset to complete
00066 }
00067
00068 bool BME280::read_raw_all(int32_t* temperature, int32_t* pressure, int32_t* humidity) {
00069     if (!initialized_) {
00070         std::cerr << "BME280 not initialized_.\n";
00071         return false;
00072     }
00073
00074     // Define the starting register address
00075     uint8_t start_reg = REG_PRESSURE_MSB;
00076     // Total bytes to read: 3 (pressure) + 3 (temperature) + 2 (humidity) = 8
00077     uint8_t buf[8] = {0};
00078
00079     // Write the starting register address
00080     int ret = i2c_write_blocking(i2c_port, device_addr, &start_reg, 1, true);
00081     if (ret != 1) {
00082         std::cerr << "Failed to write starting register address to BME280.\n";
00083         return false;
00084     }
00085
00086     // Read data
00087     ret = i2c_read_blocking(i2c_port, device_addr, buf, 8, false);
00088     if (ret != 8) {
00089         std::cerr << "Failed to read data from BME280.\n";
00090         return false;
00091     }
00092
00093     // Combine bytes to form raw values
00094     *pressure = ((int32_t)buf[0] << 12) | ((int32_t)buf[1] << 4) | ((int32_t)(buf[2] >> 4));
00095     *temperature = ((int32_t)buf[3] << 12) | ((int32_t)buf[4] << 4) | ((int32_t)(buf[5] >> 4));
00096     *humidity = ((int32_t)buf[6] << 8) | (int32_t)buf[7];
00097
00098     return true;
00099 }
00100
00101 float BME280::convert_temperature(int32_t temp_raw) const {
00102     int32_t var1, var2;
00103     var1 = (((temp_raw >> 3) - ((int32_t)calib_params.dig_t1 << 1))) * ((int32_t)calib_params.dig_t2)
00104     » 11;
00105     var2 = (((((temp_raw >> 4) - ((int32_t)calib_params.dig_t1)) * ((temp_raw >> 4) -
00106     ((int32_t)calib_params.dig_t1)) » 12) * ((int32_t)calib_params.dig_t3)) » 14;
00107     t_fine = var1 + var2;
00108     float T = (t_fine * 5 + 128) » 8;
00109     return T / 100.0f;
00110 }
00111
00112 float BME280::convert_pressure(int32_t pressure_raw) const {
00113     int64_t var1, var2, p;
00114     var1 = ((int64_t)t_fine) - 128000;
00115     var2 = var1 * var1 * (int64_t)calib_params.dig_p6;
00116     var2 = var2 + ((var1 * (int64_t)calib_params.dig_p5) » 17);
00117     var2 = var2 + (((int64_t)calib_params.dig_p4) » 35);
00118     var1 = ((var1 * var1 * (int64_t)calib_params.dig_p3) » 8) + ((var1 * (int64_t)calib_params.dig_p2)
00119     » 12);
00120     var1 = (((int64_t)1 << 47) + var1) * ((int64_t)calib_params.dig_p1) » 33;
00121
00122     if (var1 == 0) {
00123         return 0.0f; // avoid exception caused by division by zero
00124     }
00125     p = 1048576 - pressure_raw;
00126     p = (((p << 31) - var2) * 3125) / var1;
00127     var1 = (((int64_t)calib_params.dig_p9) * (p » 13) * (p » 13)) » 25;
00128     var2 = (((int64_t)calib_params.dig_p8) * p) » 19;
00129
00130     p = ((p + var1 + var2) » 8) + (((int64_t)calib_params.dig_p7) » 4);
00131     return (float)p / 25600.0f; // in hPa
00132 }
00133
00134 float BME280::convert_humidity(int32_t humidity_raw) const {
00135     int32_t v_x1_u32r;
00136     v_x1_u32r = t_fine - 76800;
00137     v_x1_u32r = (((((humidity_raw << 14) - ((int32_t)calib_params.dig_h4 << 20) -
00138     ((int32_t)calib_params.dig_h5 * v_x1_u32r) + 16384) » 15) *
00139     (((((v_x1_u32r * (int32_t)calib_params.dig_h6) » 10) * (((v_x1_u32r *
00140     (int32_t)calib_params.dig_h3) » 11) + 32768)) » 10) + 2097152) *

```

```

00136             (int32_t)calib_params.dig_h2 + 8192) >> 14));
00137     v_xl_u32r = std::max(v_xl_u32r, (int32_t)0);
00138     v_xl_u32r = std::min(v_xl_u32r, (int32_t)419430400);
00139     float h = v_xl_u32r >> 12;
00140     return h / 1024.0f;
00141 }
00142
00143 bool BME280::get_calibration_parameters() {
00144     // Read temperature and pressure calibration data (0x88 to 0xA1)
00145     uint8_t calib_data[26];
00146     uint8_t reg = REG_DIG_T1_LSB;
00147     int ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
00148     if (ret != 1) {
00149         std::cerr << "Failed to write to BME280.\n";
00150         return false;
00151     }
00152     ret = i2c_read_blocking(i2c_port, device_addr, calib_data, 26, false);
00153     if (ret != 26) {
00154         std::cerr << "Failed to read calibration data from BME280.\n";
00155         return false;
00156     }
00157
00158     // Parse temperature calibration data
00159     calib_params.dig_t1 = (uint16_t)(calib_data[1] << 8 | calib_data[0]);
00160     calib_params.dig_t2 = (int16_t)(calib_data[3] << 8 | calib_data[2]);
00161     calib_params.dig_t3 = (int16_t)(calib_data[5] << 8 | calib_data[4]);
00162
00163     // Parse pressure calibration data
00164     calib_params.dig_p1 = (uint16_t)(calib_data[7] << 8 | calib_data[6]);
00165     calib_params.dig_p2 = (int16_t)(calib_data[9] << 8 | calib_data[8]);
00166     calib_params.dig_p3 = (int16_t)(calib_data[11] << 8 | calib_data[10]);
00167     calib_params.dig_p4 = (int16_t)(calib_data[13] << 8 | calib_data[12]);
00168     calib_params.dig_p5 = (int16_t)(calib_data[15] << 8 | calib_data[14]);
00169     calib_params.dig_p6 = (int16_t)(calib_data[17] << 8 | calib_data[16]);
00170     calib_params.dig_p7 = (int16_t)(calib_data[19] << 8 | calib_data[18]);
00171     calib_params.dig_p8 = (int16_t)(calib_data[21] << 8 | calib_data[20]);
00172     calib_params.dig_p9 = (int16_t)(calib_data[23] << 8 | calib_data[22]);
00173
00174     calib_params.dig_h1 = calib_data[25];
00175
00176     // Read humidity calibration data (0xE1 to 0xE7)
00177     reg = 0xE1;
00178     ret = i2c_write_blocking(i2c_port, device_addr, &reg, 1, true);
00179     if (ret != 1) {
00180         std::cerr << "Failed to write to BME280 for humidity calibration.\n";
00181         return false;
00182     }
00183
00184     uint8_t hum_calib_data[7];
00185     ret = i2c_read_blocking(i2c_port, device_addr, hum_calib_data, 7, false);
00186     if (ret != 7) {
00187         std::cerr << "Failed to read humidity calibration data from BME280.\n";
00188         return false;
00189     }
00190
00191     // Parse humidity calibration data
00192     calib_params.dig_h2 = (int16_t)(hum_calib_data[1] << 8 | hum_calib_data[0]);
00193     calib_params.dig_h3 = hum_calib_data[2];
00194     calib_params.dig_h4 = (int16_t)((hum_calib_data[3] << 4) | (hum_calib_data[4] & 0x0F));
00195     calib_params.dig_h5 = (int16_t)((hum_calib_data[5] << 4) | (hum_calib_data[4] >> 4));
00196     calib_params.dig_h6 = (int8_t)hum_calib_data[6];
00197
00198     return true;
00199 }
00200
00201 bool BME280::configure_sensor() {
00202     uint8_t buf[2];
00203
00204     // Set humidity oversampling (must be set before ctrl_meas)
00205     buf[0] = REG_CTRL_HUM;
00206     buf[1] = 0x05; // Humidity oversampling xl6
00207     int ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00208     if (ret != 2) {
00209         std::cerr << "Failed to write CTRL_HUM to BME280.\n";
00210         return false;
00211     }
00212
00213     // Write config register
00214     buf[0] = REG_CONFIG;
00215     buf[1] = 0x00; // Default settings
00216     ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00217     if (ret != 2) {
00218         std::cerr << "Failed to write CONFIG to BME280.\n";
00219         return false;
00220     }
00221
00222     // Write ctrl_meas register

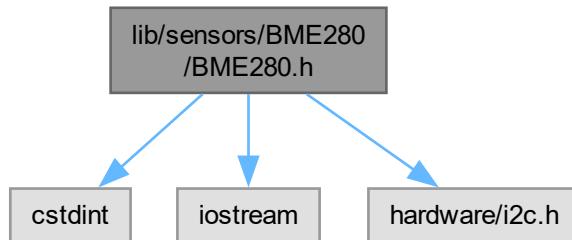
```

```

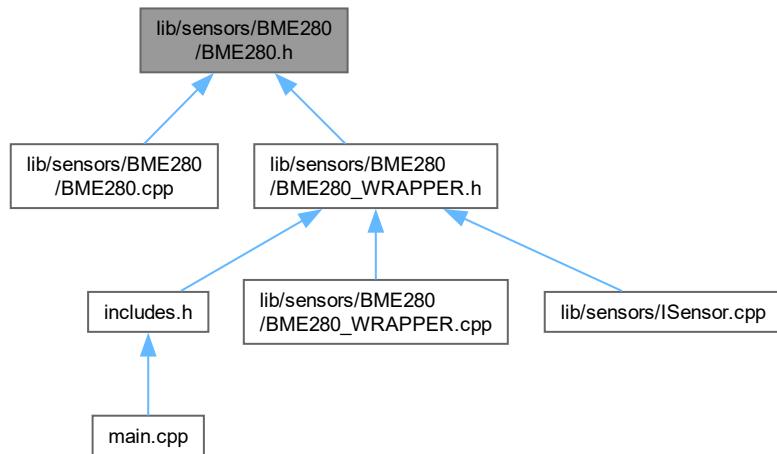
00223     buf[0] = REG_CTRL_MEAS;
00224     buf[1] = 0xB7; // Temp and pressure oversampling x16, normal mode
00225     ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00226     if (ret !=2) {
00227         std::cerr << "Failed to write CTRL_MEAS to BME280.\n";
00228         return false;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

8.85 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.](#)

```

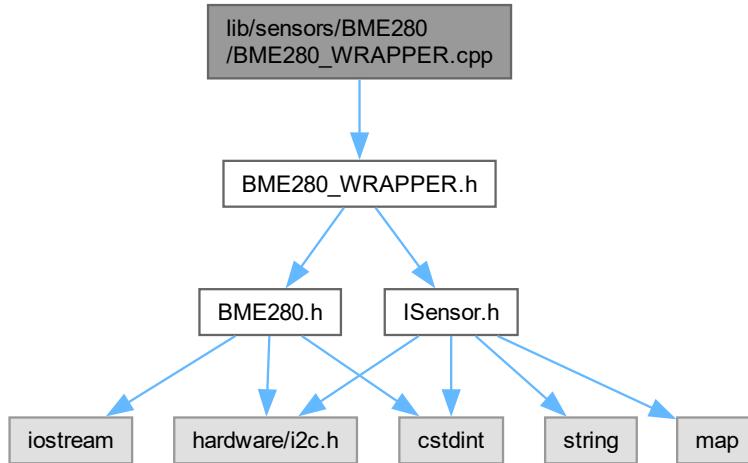
00001 // BME280.h
00002
00003 #ifndef BME280_H
00004 #define BME280_H
00005
00006 #include <cstdint>
00007 #include <iostream>
00008 #include "hardware/i2c.h"
00009
00010 // Calibration parameters structure
00011 struct BME280CalibParam {
00012     // Temperature parameters
00013     uint16_t dig_t1;
00014     int16_t dig_t2;
00015     int16_t dig_t3;
00016
00017     // Pressure parameters
00018     uint16_t dig_p1;
00019     int16_t dig_p2;
00020     int16_t dig_p3;
00021     int16_t dig_p4;
00022     int16_t dig_p5;
00023     int16_t dig_p6;
00024     int16_t dig_p7;
00025     int16_t dig_p8;
00026     int16_t dig_p9;
00027
00028     // Humidity parameters
00029     uint8_t dig_h1;
00030     int16_t dig_h2;
00031     uint8_t dig_h3;
00032     int16_t dig_h4;
00033     int16_t dig_h5;
00034     int8_t dig_h6;
00035 };
00036
00037 // BME280 Class Definition
00038 class BME280 {
00039 public:
00040     // I2C Address Options
00041     static constexpr uint8_t ADDR_SDO_LOW = 0x76;
00042     static constexpr uint8_t ADDR_SDO_HIGH = 0x77;
00043
00044     // Constructor
00045     BME280(i2c_inst_t* i2cPort, uint8_t address = ADDR_SDO_LOW);
00046
00047     // Initialize the sensor
00048     bool init();
00049
00050     // Reset the sensor
00051     void reset();
00052
00053     // Read all raw data: temperature, pressure, and humidity
00054     bool read_raw_all(int32_t* temperature, int32_t* pressure, int32_t* humidity);
00055
00056     // Convert raw data to actual values
00057     float convert_temperature(int32_t temp_raw) const;
00058     float convert_pressure(int32_t pressure_raw) const;
00059     float convert_humidity(int32_t humidity_raw) const;
00060
00061 private:
00062     // Configure the sensor
00063     bool configure_sensor();
00064
00065     // Retrieve calibration parameters from the sensor
00066     bool get_calibration_parameters();
00067
00068     // I2C port and device address
00069     i2c_inst_t* i2c_port;
00070     uint8_t device_addr;
00071

```

```
00072 // Calibration parameters
00073 BME280CalibParam calib_params;
00074
00075 // Initialization status
00076 bool initialized_;
00077
00078 // Fine temperature parameter needed for compensation
00079 mutable int32_t t_fine;
00080
00081 // Register Definitions
00082 static constexpr uint8_t REG_CONFIG = 0xF5;
00083 static constexpr uint8_t REG_CTRL_MEAS = 0xF4;
00084 static constexpr uint8_t REG_CTRL_HUM = 0xF2;
00085 static constexpr uint8_t REG_RESET = 0xE0;
00086
00087 static constexpr uint8_t REG_PRESSURE_MSB = 0xF7;
00088 static constexpr uint8_t REG_TEMPERATURE_MSB = 0xFA;
00089 static constexpr uint8_t REG_HUMIDITY_MSB = 0xFD;
00090
00091 // Calibration Registers
00092 static constexpr uint8_t REG_DIG_T1_LSB = 0x88;
00093 static constexpr uint8_t REG_DIG_T1_MSB = 0x89;
00094 static constexpr uint8_t REG_DIG_T2_LSB = 0x8A;
00095 static constexpr uint8_t REG_DIG_T2_MSB = 0x8B;
00096 static constexpr uint8_t REG_DIG_T3_LSB = 0x8C;
00097 static constexpr uint8_t REG_DIG_T3_MSB = 0x8D;
00098
00099 static constexpr uint8_t REG_DIG_P1_LSB = 0x8E;
00100 static constexpr uint8_t REG_DIG_P1_MSB = 0x8F;
00101 static constexpr uint8_t REG_DIG_P2_LSB = 0x90;
00102 static constexpr uint8_t REG_DIG_P2_MSB = 0x91;
00103 static constexpr uint8_t REG_DIG_P3_LSB = 0x92;
00104 static constexpr uint8_t REG_DIG_P3_MSB = 0x93;
00105 static constexpr uint8_t REG_DIG_P4_LSB = 0x94;
00106 static constexpr uint8_t REG_DIG_P4_MSB = 0x95;
00107 static constexpr uint8_t REG_DIG_P5_LSB = 0x96;
00108 static constexpr uint8_t REG_DIG_P5_MSB = 0x97;
00109 static constexpr uint8_t REG_DIG_P6_LSB = 0x98;
00110 static constexpr uint8_t REG_DIG_P6_MSB = 0x99;
00111 static constexpr uint8_t REG_DIG_P7_LSB = 0x9A;
00112 static constexpr uint8_t REG_DIG_P7_MSB = 0x9B;
00113 static constexpr uint8_t REG_DIG_P8_LSB = 0x9C;
00114 static constexpr uint8_t REG_DIG_P8_MSB = 0x9D;
00115 static constexpr uint8_t REG_DIG_P9_LSB = 0x9E;
00116 static constexpr uint8_t REG_DIG_P9_MSB = 0x9F;
00117
00118 // Humidity Calibration Registers
00119 static constexpr uint8_t REG_DIG_H1 = 0xA1;
00120 static constexpr uint8_t REG_DIG_H2 = 0xE1;
00121 static constexpr uint8_t REG_DIG_H3 = 0xE3;
00122 static constexpr uint8_t REG_DIG_H4 = 0xE4;
00123 static constexpr uint8_t REG_DIG_H5 = 0xE5;
00124 static constexpr uint8_t REG_DIG_H6 = 0xE7;
00125
00126 // Number of calibration parameters to read
00127 static constexpr size_t NUM_CALIB_PARAMS = 24;
00128 };
00129
00130 #endif // BME280_H
```

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

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



8.88 BME280_WRAPPER.cpp

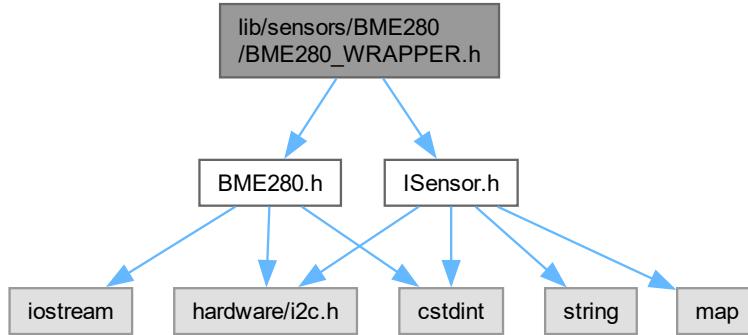
[Go to the documentation of this file.](#)

```

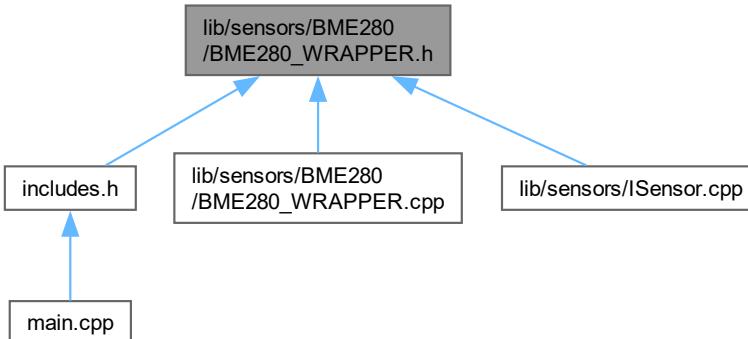
00001 #include "BME280_WRAPPER.h"
00002
00003 BME280Wrapper::BME280Wrapper(i2c_inst_t* i2c) : sensor_(i2c) {}
00004
00005 bool BME280Wrapper::init() {
00006     initialized_ = sensor_.init();
00007     return initialized_;
00008 }
00009
00010 float BME280Wrapper::read_data(SensorDataTypeIdentifier type) {
00011     int32_t temp_raw, pressure_raw, humidity_raw;
00012     sensor_.read_raw_all(&temp_raw, &pressure_raw, &humidity_raw);
00013
00014     switch(type) {
00015         case SensorDataTypeIdentifier::TEMPERATURE:
00016             return sensor_.convert_temperature(temp_raw);
00017         case SensorDataTypeIdentifier::PRESSURE:
00018             return sensor_.convert_pressure(pressure_raw);
00019         case SensorDataTypeIdentifier::HUMIDITY:
00020             return sensor_.convert_humidity(humidity_raw);
00021         default:
00022             return 0.0f;
00023     }
00024 }
00025
00026 bool BME280Wrapper::is_initialized() const {
00027     return initialized_;
00028 }
00029
00030 SensorType BME280Wrapper::get_type() const {
00031     return SensorType::ENVIRONMENT;
00032 }
00033
00034 bool BME280Wrapper::configure(const std::map<std::string, std::string>& config) {
00035     return true;
00036 }
```

8.89 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
```

```

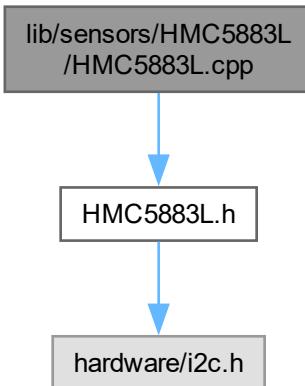
00003 #define BME280_WRAPPER_H
00004
00005 #include "ISensor.h"
00006 #include "BME280.h"
00007
00008 class BME280Wrapper : public ISensor {
00009 private:
0010     BME280 sensor_;
0011     bool initialized_ = false;
0012
0013 public:
0014     BME280Wrapper(i2c_inst_t* i2c);
0015
0016     bool init() override;
0017     float read_data(SensorDataTypeIdentifier type) override;
0018     bool is_initialized() const override;
0019     SensorType get_type() const override;
0020     bool configure(const std::map<std::string, std::string>& config) override;
0021
0022 };
0023
0024 #endif // BME280_WRAPPER_H

```

8.91 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.](#)

```

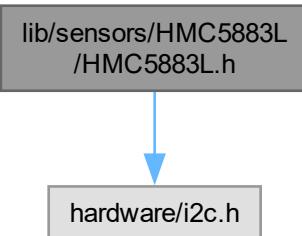
00001 #include "HMC5883L.h"
00002
00003 HMC5883L::HMC5883L(i2c_inst_t* i2c, uint8_t address) : i2c(i2c), address(address) {}
00004
00005 bool HMC5883L::init() {
00006     // Continuous measurement mode, 15Hz data output rate
00007     if (!write_register(0x00, 0x70)) return false;
00008     if (!write_register(0x01,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

```

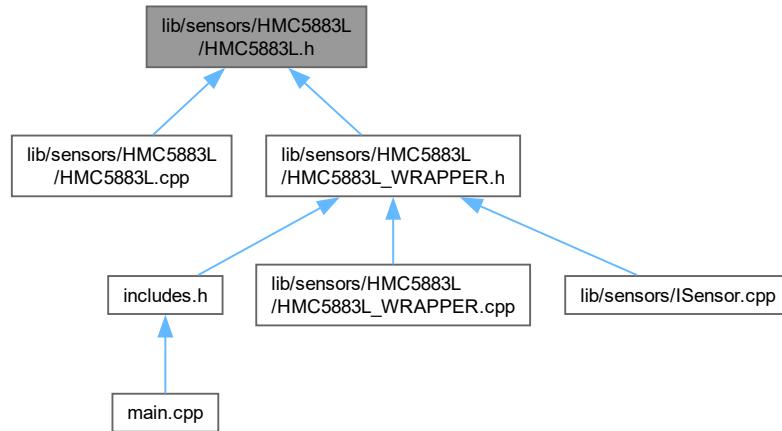
```
00013 bool HMC5883L::read(int16_t& x, int16_t& y, int16_t& z) {
00014     uint8_t buffer[6];
00015     if (!read_register(0x03, buffer, 6)) return false;
00016
00017     x = (buffer[0] << 8) | buffer[1];
00018     z = (buffer[2] << 8) | buffer[3];
00019     y = (buffer[4] << 8) | buffer[5];
00020
00021     if (x > 32767) x -= 65536;
00022     if (y > 32767) y -= 65536;
00023     if (z > 32767) z -= 65536;
00024
00025     return true;
00026 }
00027
00028 bool HMC5883L::write_register(uint8_t reg, uint8_t value) {
00029     uint8_t buffer[2] = {reg, value};
00030     return i2c_write_blocking(i2c, address, buffer, 2, false) == 2;
00031 }
00032
00033 bool HMC5883L::read_register(uint8_t reg, uint8_t* buffer, size_t length) {
00034     if (i2c_write_blocking(i2c, address, &reg, 1, true) != 1) return false;
00035     return i2c_read_blocking(i2c, address, buffer, length, false) == length;
00036 }
```

8.93 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.](#)

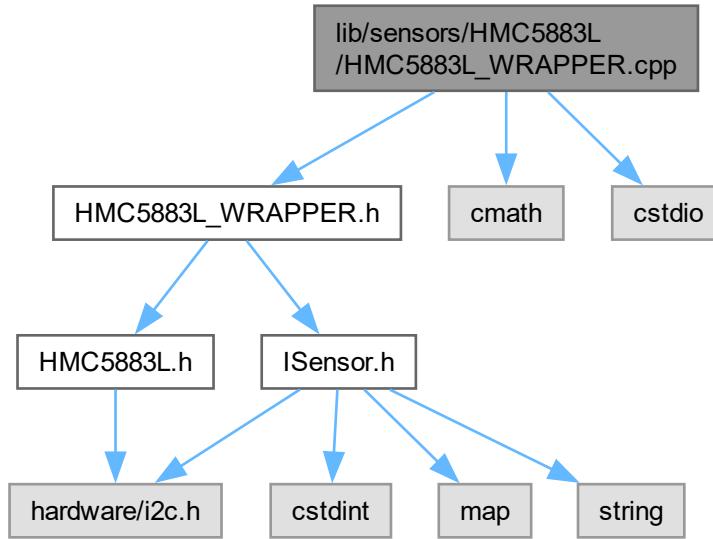
```

00001 #ifndef HMC5883L_H
00002 #define HMC5883L_H
00003
00004 #include "hardware/i2c.h"
00005
00006 class HMC5883L {
00007 public:
00008     HMC5883L(i2c_inst_t* i2c, uint8_t address = 0x0D);
00009     bool init();
00010     bool read(int16_t& x, int16_t& y, int16_t& z);
00011
00012 private:
00013     i2c_inst_t* i2c;
00014     uint8_t address;
00015
00016     bool write_register(uint8_t reg, uint8_t value);
00017     bool read_register(uint8_t reg, uint8_t* buffer, size_t length);
00018 };
00019
00020 #endif
  
```

8.95 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.](#)

```

00001 #include "HMC5883L_WRAPPER.h"
00002 #include <cmath>
00003 #include <cstdio>
00004
00005 HMC5883LWrapper::HMC5883LWrapper(i2c_inst_t* i2c) : sensor_(i2c), initialized_(false) {}
00006
00007 bool HMC5883LWrapper::init() {
00008     initialized_ = sensor_.init();
00009     return initialized_;
00010 }
00011
00012 float HMC5883LWrapper::read_data(SensorDataTypeIdentifier type) {
00013     if (!initialized_) return 0.0f;
00014
00015     int16_t x, y, z;
00016     if (!sensor_.read(x, y, z)) return 0.0f;
00017
00018     const float LSB_TO_UT = 100.0 / 1090.0;
00019     float x_uT = x * LSB_TO_UT;
00020     float y_uT = y * LSB_TO_UT;
00021     float z_uT = z * LSB_TO_UT;
00022
00023     switch (type) {
00024         case SensorDataTypeIdentifier::MAG_FIELD_X:
00025             return x_uT;
00026         case SensorDataTypeIdentifier::MAG_FIELD_Y:
00027             return y_uT;
00028         case SensorDataTypeIdentifier::MAG_FIELD_Z:
00029             return z_uT;
00030         default:
00031             return 0.0f;
00032     }
00033 }
00034
00035 bool HMC5883LWrapper::is_initialized() const {
00036     return initialized_;
  
```

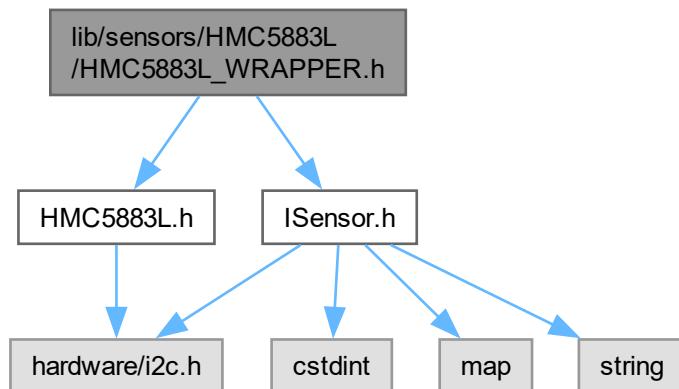
```

00037 }
00038
00039 SensorType HMC5883LWrapper::get_type() const {
00040     return SensorType::MAGNETOMETER;
00041 }
00042
00043 bool HMC5883LWrapper::configure(const std::map<std::string, std::string>& config) {
00044     // Configuration logic if needed
00045     return true;
00046 }

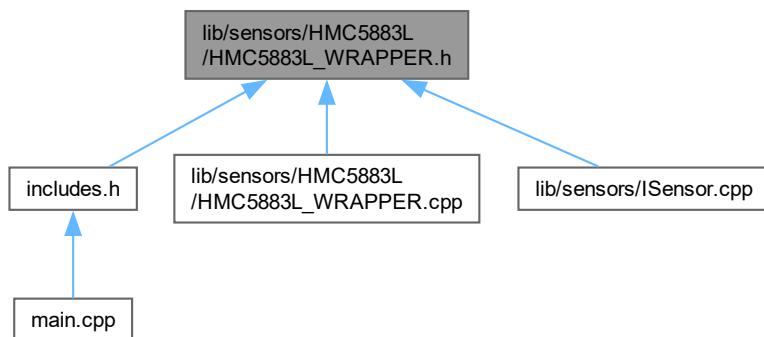
```

8.97 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

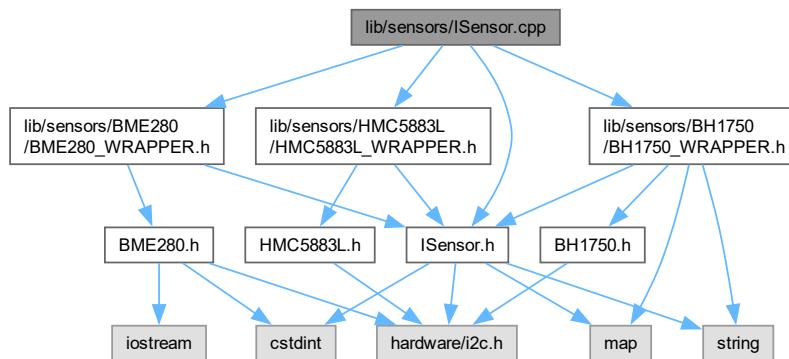
[Go to the documentation of this file.](#)

```
00001 #ifndef HMC5883L_WRAPPER_H
00002 #define HMC5883L_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "HMC5883L.h"
00006
00007 class HMC5883LWrapper : public ISensor {
00008 public:
00009     HMC5883LWrapper(i2c_inst_t* i2c);
00010     bool init() override;
00011     float read_data(SensorDataTypeIdentifier type) override;
00012     bool is_initialized() const override;
00013     SensorType get_type() const override;
00014     bool configure(const std::map<std::string, std::string>& config) override;
00015
00016 private:
00017     HMC5883L sensor_;
00018     bool initialized_;
00019 };
00020
00021 #endif
```

8.99 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

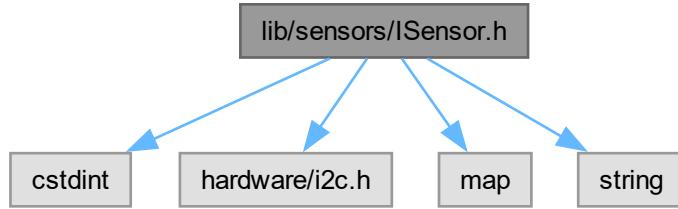
[Go to the documentation of this file.](#)

```
00001 // ISensor.cpp
00002 #include "ISensor.h"
00003 #include "lib/sensors/BH1750/BH1750_WRAPPER.h"
00004 #include "lib/sensors/BME280/BME280_WRAPPER.h"
00005 #include "lib/sensors/HMC5883L/HMC5883L_WRAPPER.h"
00006
00013
00018
00023 SensorWrapper& SensorWrapper::get_instance() {
00024     static SensorWrapper instance;
00025     return instance;
00026 }
00027
00031 SensorWrapper::SensorWrapper() = default;
00032
00039 bool SensorWrapper::sensor_init(SensorType type, i2c_inst_t* i2c) {
00040     switch(type) {
00041         case SensorType::LIGHT:
00042             sensors[type] = new BH1750Wrapper();
00043             break;
00044         case SensorType::ENVIRONMENT:
00045             sensors[type] = new BME280Wrapper(i2c);
00046             break;
00047         case SensorType::IMU:
00048             //sensors[type] = new MPU6050Wrapper(i2c);
00049             break;
00050         case SensorType::MAGNETOMETER:
00051             sensors[type] = new HMC5883LWrapper(i2c);
00052             break;
00053     }
00054     return sensors[type]->init();
00055 }
00056
00063 bool SensorWrapper::sensor_configure(SensorType type, const std::map<std::string, std::string>& config) {
00064     auto it = sensors.find(type);
00065     if (it != sensors.end() && it->second->is_initialized()) {
00066         return it->second->configure(config);
00067     }
00068     std::cerr << "Sensor not initialized or not found: " << static_cast<int>(type) << std::endl;
00069     return false;
00070 }
00071
00078 float SensorWrapper::sensor_read_data(SensorType sensorType, SensorDataTypeIdentifier dataType) {
00079     auto it = sensors.find(sensorType);
00080     if (it != sensors.end() && it->second->is_initialized()) {
00081         return it->second->read_data(dataType);
00082     }
00083     return 0.0f;
00084 }
```

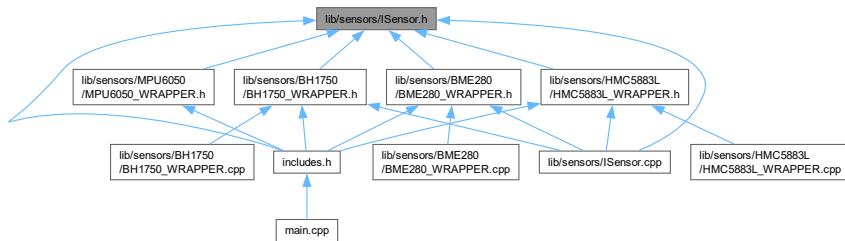
8.101 lib/sensors/ISensor.h File Reference

```
#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 [SensorDataTypeldentifier](#) {
`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,
```

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

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

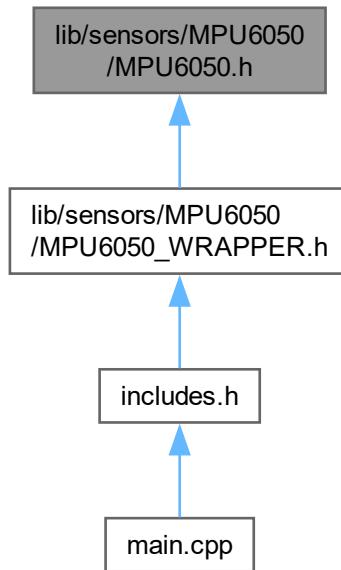
8.104 MPU6050.cpp

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

00001

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

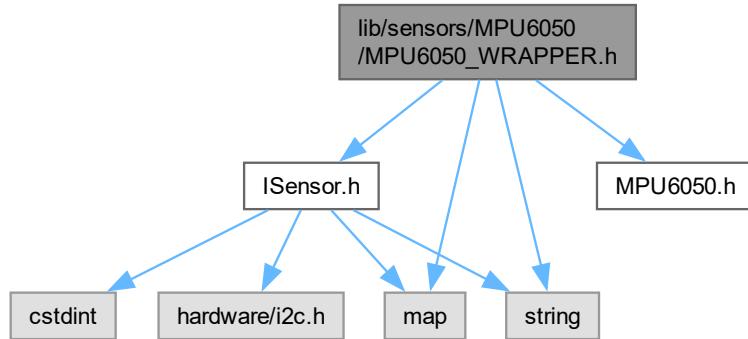
[Go to the documentation of this file.](#)

00001

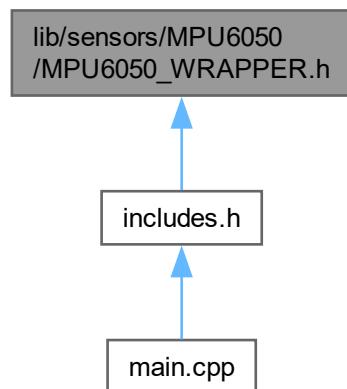
8.109 lib/sensors/MPU6050/MPU6050_WRAPPER.h File Reference

```
#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
```

```

00004 #include "ISensor.h"
00005 #include "MPU6050.h"
00006 #include <map>
00007 #include <string>
00008
00009 class MPU6050Wrapper : public ISensor {
00010 private:
00011     MPU6050 sensor_;
00012     bool initialized_ = false;
00013
00014 public:
00015     MPU6050Wrapper();
00016
00017     bool init() override;
00018     float read_data(SensorDataTypeIdentifier type) override;
00019     bool is_initialized() const override;
00020     SensorType get_type() const override;
00021
00022     bool configure(const std::map<std::string, std::string>& config);
00023 };
00024
00025 #endif

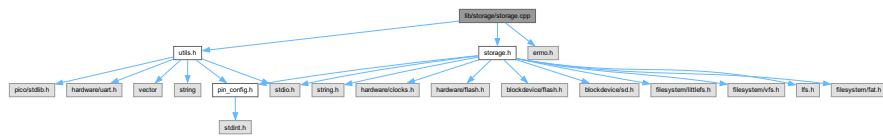
```

8.111 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.

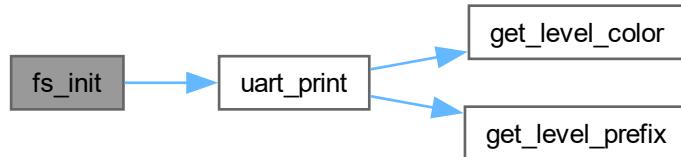
Returns

True if initialization was successful, false otherwise.

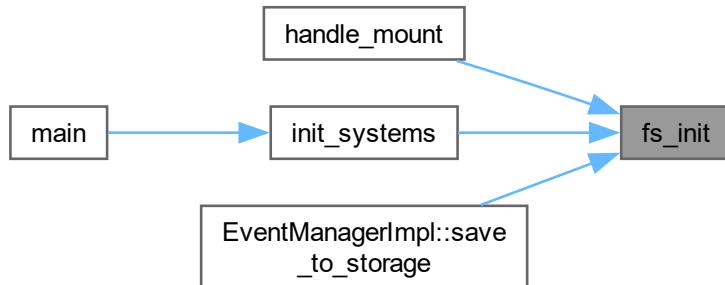
Mounts the littlefs file system on the SD card. If mounting fails, it formats the SD card with littlefs and then attempts to mount again.

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

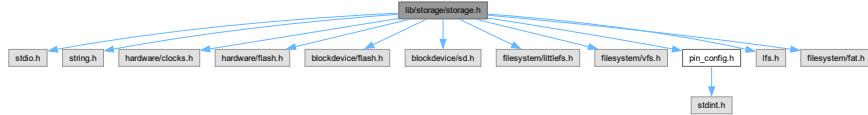
[Go to the documentation of this file.](#)

```
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 }
```

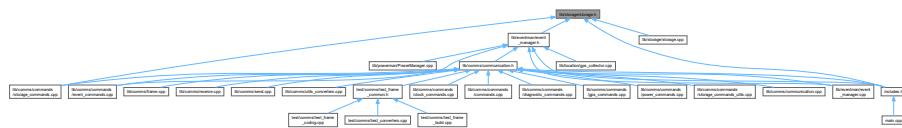
8.113 lib/storage/storage.h File Reference

```
#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.

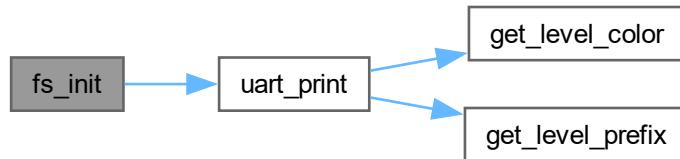
Returns

True if initialization was successful, false otherwise.

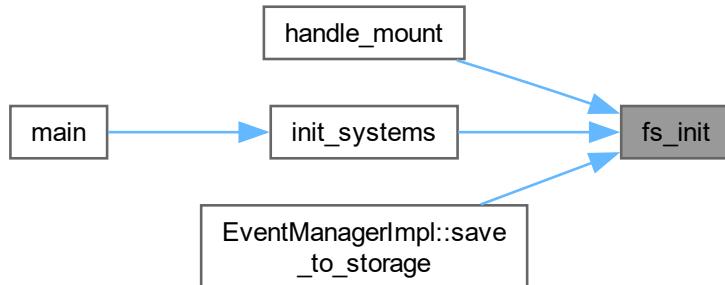
Mounts the littlefs file system on the SD card. If mounting fails, it formats the SD card with littlefs and then attempts to mount again.

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
```

```

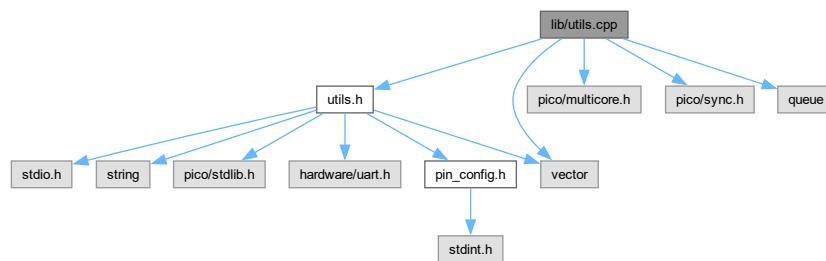
00032
00033 // void example_file_operations() {
00034 //     // Open a file for writing
00035 //     FileHandle log_file = fs_open_file("/log.txt", "w");
00036 //     if (!log_file.is_open) {
00037 //         uartPrint("Failed to open log file");
00038 //         return;
00039 //     }
00040
00041 //     // Write some data
00042 //     const char* message = "Hello, World!\n";
00043 //     ssize_t written = fs_write_file(log_file, message, strlen(message));
00044 //     if (written < 0) {
00045 //         uartPrint("Failed to write to log file");
00046 //     }
00047
00048 //     // Close the file
00049 //     fs_close_file(log_file);
00050
00051 //     // Open file for reading
00052 //     log_file = fs_open_file("/log.txt", "r");
00053 //     if (!log_file.is_open) {
00054 //         uartPrint("Failed to open log file for reading");
00055 //         return;
00056 //     }
00057
00058 //     // Read the data
00059 //     char buffer[128];
00060 //     ssize_t bytes_read = fs_read_file(log_file, buffer, sizeof(buffer) - 1);
00061 //     if (bytes_read > 0) {
00062 //         buffer[bytes_read] = '\0'; // Null terminate the string
00063 //         uartPrint(buffer);
00064 //     }
00065
00066 //     // Close the file
00067 //     fs_close_file(log_file);
00068 // }

```

8.115 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

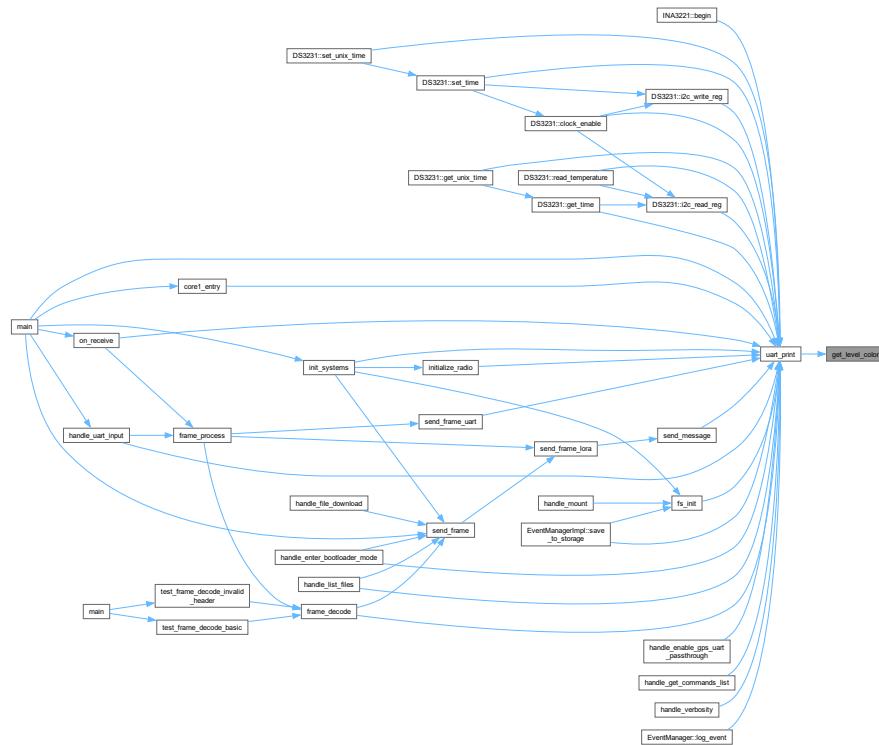
<code>level</code>	The verbosity level
--------------------	---------------------

Returns

ANSI color escape sequence

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

Here is the caller graph for this function:



8.115.2.2 `get_level_prefix()`

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

Gets text prefix for verbosity level.

Parameters

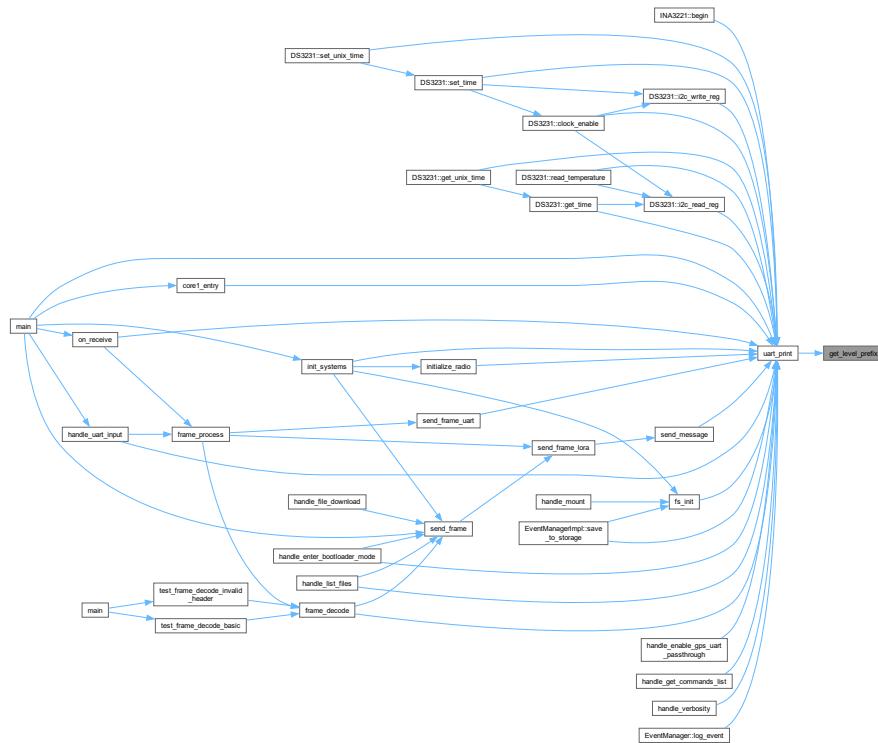
<code>level</code>	The verbosity level
--------------------	---------------------

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.

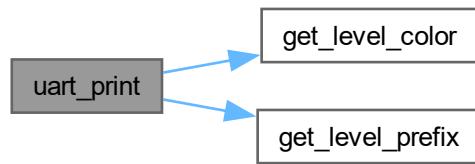
Parameters

<i>msg</i>	The message to print.
<i>logToFile</i>	A flag indicating whether to log the message to a file (currently not implemented).
<i>uart</i>	The UART instance to use for printing.

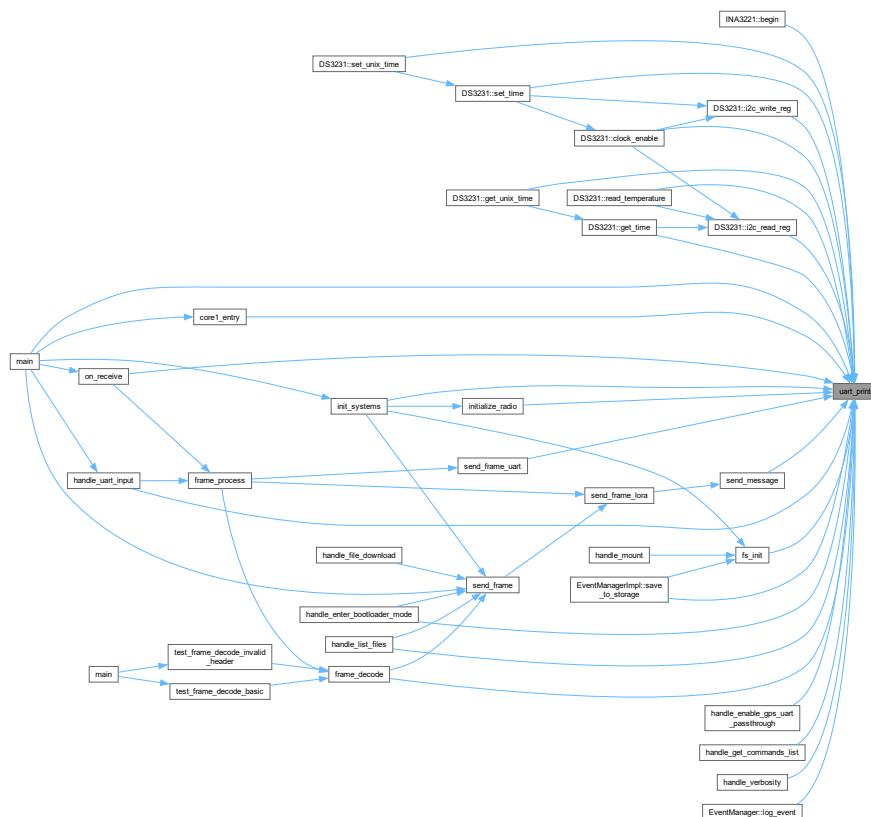
Prints the given message to the specified UART, prepending it with a timestamp and the core number. Uses a mutex to ensure thread-safe access to the UART.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.115.2.4 crc16()

```

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

Calculates the CRC16 checksum of the given data.

Calculates CRC16 checksum.

Parameters

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

Returns

The CRC16 checksum.

Calculates the CRC16 checksum using the standard algorithm.

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

8.115.3 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;
```

```

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

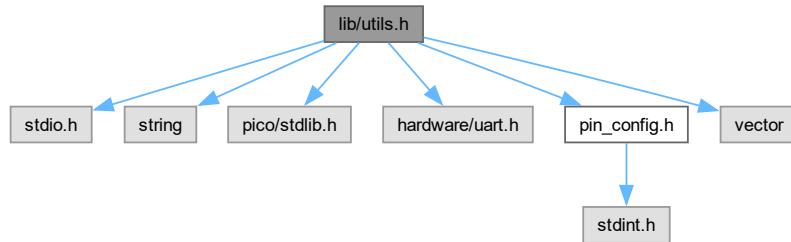
```

8.117 lib/utils.h File Reference

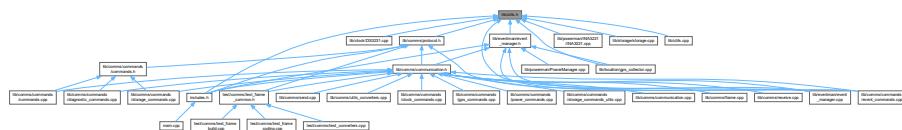
Utility functions and definitions for the Kabisat firmware.

```
#include <stdio.h>
#include <string>
#include "pico/stl.h"
#include "hardware/uart.h"
#include "pin_config.h"
#include <vector>
```

Include dependency graph for utils.h:



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



Macros

- `#define ANSI_RED "\033[31m"`
ANSI escape codes for terminal color output.
- `#define ANSI_GREEN "\033[32m"`
- `#define ANSI_YELLOW "\033[33m"`
- `#define ANSI_BLUE "\033[34m"`
- `#define ANSI_CYAN "\033[36m"`
- `#define ANSI_RESET "\033[0m"`

Enumerations

- enum class `VerbosityLevel` {
 `SILENT = 0` , `ERROR = 1` , `WARNING = 2` , `INFO = 3` ,
 `DEBUG = 4` , `EVENT = 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 Kabisat 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
DEBUG	Detailed debug information
EVENT	Events

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

<i>msg</i>	The message to print
<i>level</i>	Message verbosity level
<i>logToFile</i>	Whether to store the message in log storage
<i>uart</i>	The UART port to use

Prints a message to UART with timestamp and formatting.

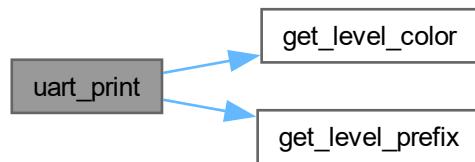
Parameters

<i>msg</i>	The message to print.
<i>logToFile</i>	A flag indicating whether to log the message to a file (currently not implemented).
<i>uart</i>	The UART instance to use for printing.

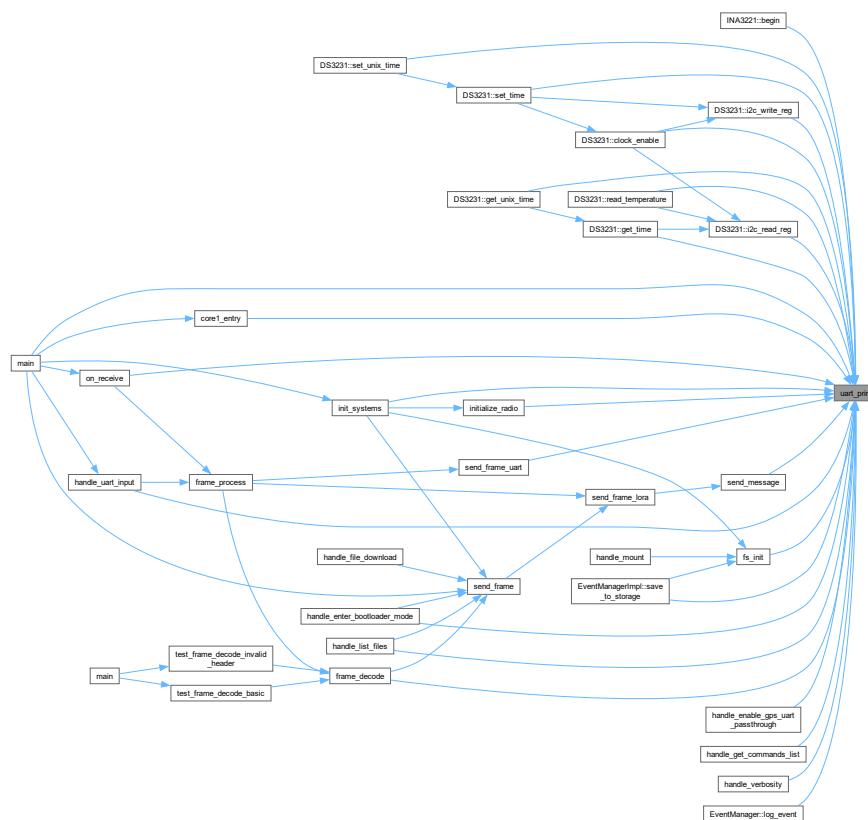
Prints the given message to the specified UART, prepending it with a timestamp and the core number. Uses a mutex to ensure thread-safe access to the UART.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.117.4.2 crc16()

```

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

Calculates CRC16 checksum.

Parameters

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

Returns

Calculated CRC16 value

Calculates CRC16 checksum.

Parameters

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

Returns

The CRC16 checksum.

Calculates the CRC16 checksum using the standard algorithm.

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

8.117.5 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.](#)

```

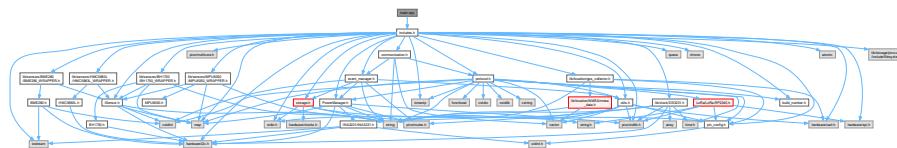
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003
00004 #include <stdio.h>
00005 #include <string>
00006 #include "pico/stdlib.h"
00007 #include "hardware/uart.h"
00008 #include "pin_config.h"
00009 #include <vector>
00010
00011
00017
00018
00019 #define ANSI_RED      "\033[31m"
00020 #define ANSI_GREEN    "\033[32m"
00021 #define ANSI_YELLOW   "\033[33m"
00022 #define ANSI_BLUE     "\033[34m"
00023 #define ANSI_CYAN     "\033[36m"
00024 #define ANSI_RESET    "\033[0m"
00025
00026
00027
00028 enum class VerbosityLevel {
00029     SILENT = 0,
00030     ERROR = 1,
00031     WARNING = 2,
00032     INFO = 3,
00033     DEBUG = 4,
00034     EVENT = 5
00035 };
00036
00037
00038 };
00039
00040
00041 extern VerbosityLevel g_uart_verbosity;
00042
00043
00044 void uart_print(const std::string& msg,
00045                  VerbosityLevel level = VerbosityLevel::INFO,
00046                  bool logToFile = false,
00047                  uart_inst_t* uart = DEBUG_UART_PORT);
00048
00049
00050
00051
00052
00053
00054
00055
00056
00057
00058
00059
00060
00061
00062
00063
00064
00065
00066
00067
00068
00069
00070
00071
00072
00073
00074
00075
00076 uint16_t crc16(const uint8_t *data, size_t length);
00077
00078 #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](#).

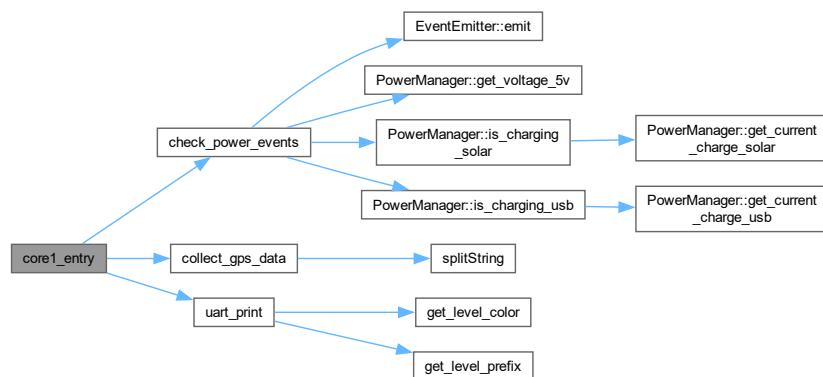
8.119.2 Function Documentation

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:

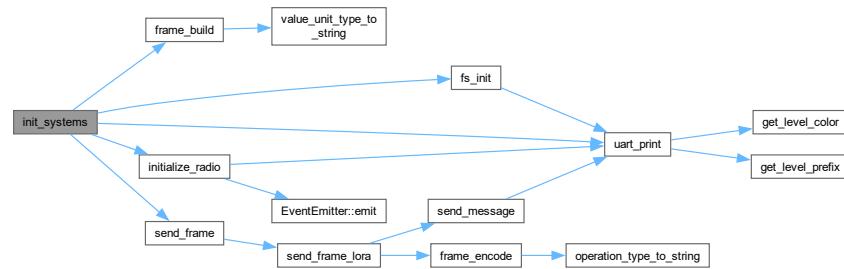


8.119.2.2 init_systems()

```
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:

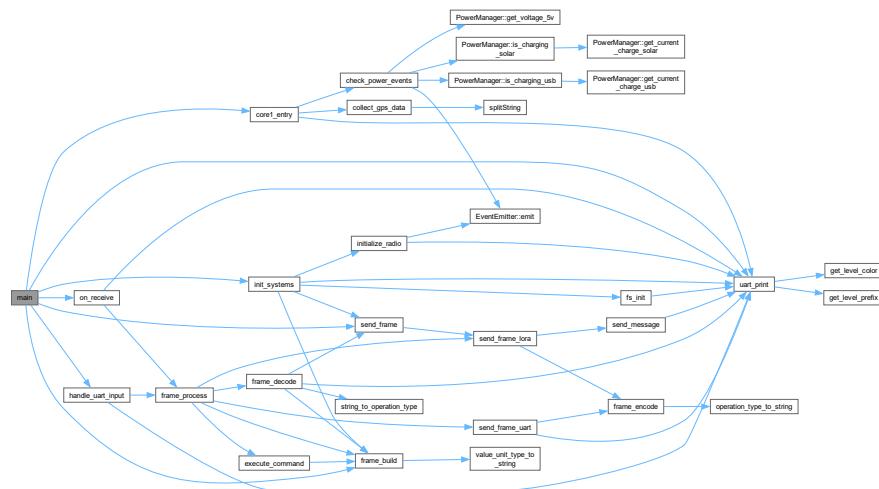


8.119.2.3 main()

```
int main (
    void )
```

Definition at line 91 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.](#)

```

00001 #include "includes.h"
00002
00003 #define LOG_FILENAME "/log.txt"
00004
00005 PowerManager powerManager(MAIN_I2C_PORT);
00006 DS3231 systemClock(MAIN_I2C_PORT);
00007
00008 char buffer[BUFFER_SIZE];
00009 int bufferIndex = 0;
0010
0011 void core1_entry() {
0012     uart_print("Starting core 1", VerbosityLevel::DEBUG);
0013     while (true) {
0014         collect_gps_data();
0015         check_power_events(powerManager);
0016         sleep_ms(100);
0017     }
0018 }
0019
0020 bool init_systems() {
0021     stdio_init_all();
0022
0023     uart_init(DEBUG_UART_PORT, DEBUG_UART_BAUD_RATE);
0024     gpio_set_function(DEBUG_UART_TX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_TX_PIN));
0025     gpio_set_function(DEBUG_UART_RX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_RX_PIN));
0026
0027     gpio_init(PICO_DEFAULT_LED_PIN);
0028     gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
0029
0030     i2c_init(MAIN_I2C_PORT, 400 * 1000);
0031     gpio_set_function(MAIN_I2C_SCL_PIN, GPIO_FUNC_I2C);
0032     gpio_set_function(MAIN_I2C_SDA_PIN, GPIO_FUNC_I2C);
0033     gpio_pull_up(MAIN_I2C_SCL_PIN);
0034     gpio_pull_up(MAIN_I2C_SDA_PIN);
0035
0036     uart_init(GPS_UART_PORT, GPS_UART_BAUD_RATE);
0037     gpio_set_function(GPS_UART_TX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_TX_PIN));
0038     gpio_set_function(GPS_UART_RX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_RX_PIN));
0039
0040     if (true)
0041     {
0042         gpio_init(GPS_POWER_ENABLE_PIN);
0043         gpio_set_dir(GPS_POWER_ENABLE_PIN, GPIO_OUT);
0044         gpio_put(GPS_POWER_ENABLE_PIN, 1);
0045     }
0046     system("color");
0047
0048     bool radioInitSuccess = false;
0049
0050     radioInitSuccess = initialize_radio();
0051
0052     // TODO: everything runs properly without errors, but file does not exist on sd card after reading
0053     // with usb reader
0054     bool sdInitDone = fs_init();
0055     if (sdInitDone) {
0056         FILE *fp = fopen(LOG_FILENAME, "w");
0057         if (fp) {
0058             uart_print("Log file opened.");
0059             int bytesWritten = fprintf(fp, "System init started.\n");
0060             uart_print("Written " + std::to_string(bytesWritten) + " bytes.");
0061             int closeStatus = fclose(fp);
0062             uart_print("Close file status: " + std::to_string(closeStatus));
0063
0064             // Get file size
0065             struct stat file_stat;
0066             if (stat(LOG_FILENAME, &file_stat) == 0) {
0067                 size_t fileSize = file_stat.st_size;
0068                 uart_print("File size: " + std::to_string(fileSize) + " bytes");
0069             } else {
0070                 uart_print("Failed to get file size");
0071             }
0072
0073             // Print file path (assuming it's in the root directory)
0074             uart_print("File path: /" + std::string(LOG_FILENAME));
0075         } else {
0076             uart_print("Failed to open log file for writing.");
0077         }
0078
0079         uart_print("SD card init: " + std::to_string(sdInitDone));
0080         std::string bootString = "System init completed @ " +
0081             std::to_string(to_ms_since_boot(get_absolute_time())) + " ms";

```

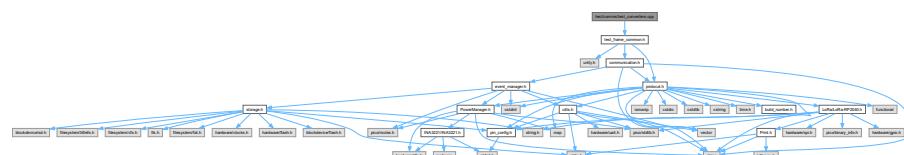
```

00081     uart_print(bootString);
00082
00083     Frame boot = frame_build(ExecutionResult::INFO, 0, 0, "HELLO");
00084     send_frame(boot);
00085
00086     uart_print("System init done.");
00087     return radioInitSuccess;
00088 }
00089
00090
00091 int main()
00092 {
00093     init_systems();
00094     multicore_launch_core1(core1_entry);
00095
00096     gpio_put(PICO_DEFAULT_LED_PIN, 0);
00097     bool powerManagerInitStatus = powerManager.initialize();
00098     if (powerManagerInitStatus)
00099     {
00100         std::map<std::string, std::string> powerConfig = {
00101             {"operating_mode", "continuous"},
00102             {"averaging_mode", "16"}, };
00103         powerManager.configure(powerConfig);
00104     } else {
00105         uart_print("Power manager init error");
00106     }
00107
00108     gpio_put(PICO_DEFAULT_LED_PIN, 1);
00109
00110
00111     for (int i = 5; i > 0; --i)
00112     {
00113         std::string intro = "Main loop starts in " + std::to_string(i) + " seconds...";
00114         uart_print(intro);
00115         gpio_put(PICO_DEFAULT_LED_PIN, (i%2==0));
00116         sleep_ms(1000);
00117     }
00118     gpio_put(PICO_DEFAULT_LED_PIN, 1);
00119
00120     Frame boot = frame_build(ExecutionResult::INFO, 0, 0, "START");
00121     send_frame(boot);
00122
00123     while (true)
00124     {
00125         int packetSize = LoRa.parse_packet();
00126         if (packetSize)
00127         {
00128             on_receive(packetSize);
00129         }
00130
00131         handle_uart_input();
00132     }
00133
00134     return 0;
00135 }
00136
00137 // BH1750    0X23
00138 // INA3221   0X40
00139 // BME280    0X76
00140 // 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 \(\)](#)

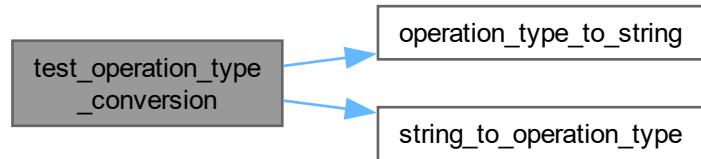
8.121.1 Function Documentation

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:

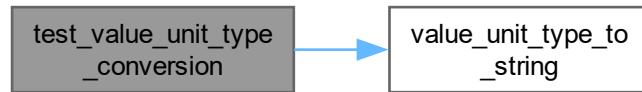


8.121.1.2 test_value_unit_type_conversion()

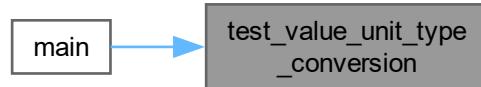
```
void test_value_unit_type_conversion (
    void )
```

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:

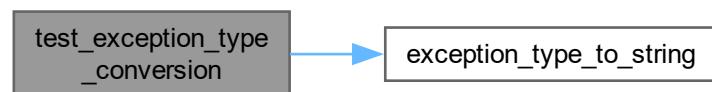


8.121.1.3 test_exception_type_conversion()

```
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()`

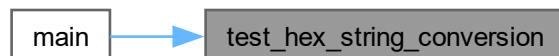
```
void test_hex_string_conversion (
    void )
```

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);
```

```

00007     OperationType converted = string_to_operation_type(str);
00008
00009     TEST_ASSERT_EQUAL(type, converted);
00010     TEST_ASSERT_EQUAL_STRING("GET", str.c_str());
00011 }
00012
0013 void test_value_unit_type_conversion() {
0014     ValueUnit unit = ValueUnit::VOLT;
0015     std::string str = value_unit_type_to_string(unit);
0016
0017     TEST_ASSERT_EQUAL_STRING("V", str.c_str());
0018 }
0019
0020 void test_exception_type_conversion() {
0021     ExceptionType type = ExceptionType::INVALID_PARAM;
0022     std::string str = exception_type_to_string(type);
0023
0024     TEST_ASSERT_EQUAL_STRING("INVALID PARAM", str.c_str());
0025 }
0026
0027 void test_hex_string_conversion() {
0028     std::string hex = "0A0B0C";
0029     std::vector<uint8_t> bytes = hex_string_to_bytes(hex);
0030
0031     TEST_ASSERT_EQUAL(3, bytes.size());
0032     TEST_ASSERT_EQUAL(0x0A, bytes[0]);
0033     TEST_ASSERT_EQUAL(0x0B, bytes[1]);
0034     TEST_ASSERT_EQUAL(0x0C, bytes[2]);
0035 }

```

8.123 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()`

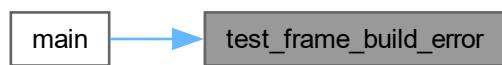
```
void test_frame_build_error (
    void )
```

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 \(\)](#)

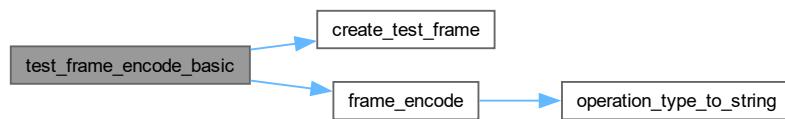
8.125.1 Function Documentation

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:

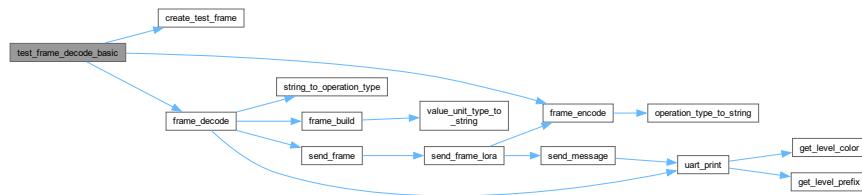


8.125.1.2 test_frame_decode_basic()

```
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:

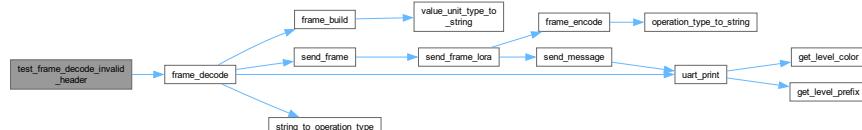


8.125.1.3 test_frame_decode_invalid_header()

```
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.](#)

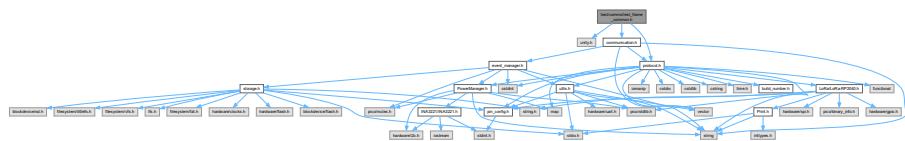
```

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 }
```

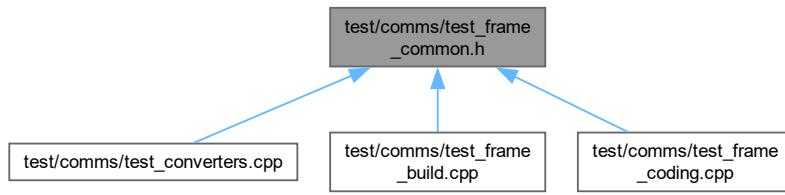
8.127 test/comms/test_frame_common.h File Reference

```
#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 ()`

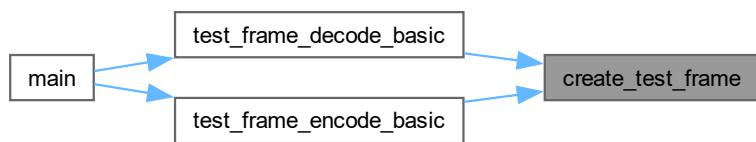
8.127.1 Function Documentation

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:



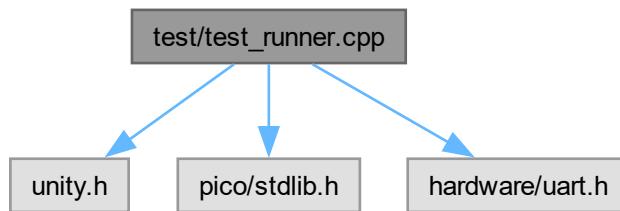
8.128 test_frame_common.h

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

```
00001 // test_frame_common.h
00002 #ifndef TEST_FRAME_COMMON_H
00003 #define TEST_FRAME_COMMON_H
00004
00005 #include "unity.h"
00006 #include "protocol.h"
00007 #include "communication.h"
00008
00009 // Helper function to create a test frame
0010 Frame create_test_frame() {
0011     Frame frame;
0012     frame.header = FRAME_BEGIN;
0013     frame.direction = 1;
0014     frame.operationType = OperationType::GET;
0015     frame.group = 1;
0016     frame.command = 2;
0017     frame.value = "test_value";
0018     frame.unit = "V";
0019     frame.footer = FRAME_END;
0020     return frame;
0021 }
0022
0023 #endif
```

8.129 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)

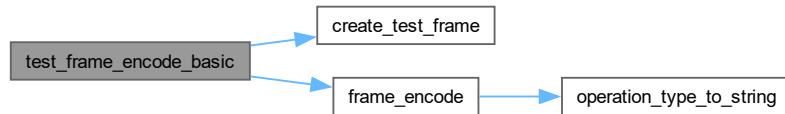
8.129.1 Function Documentation

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:

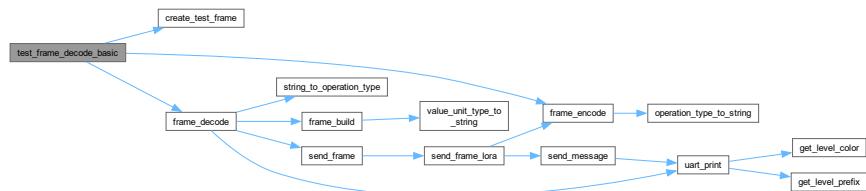


8.129.1.2 test_frame_decode_basic()

```
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:

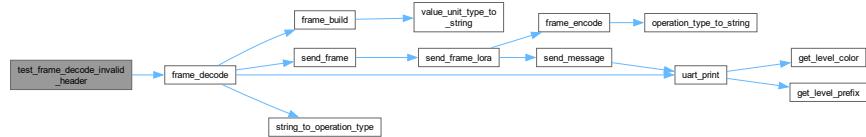


8.129.1.3 test_frame_decode_invalid_header()

```
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:

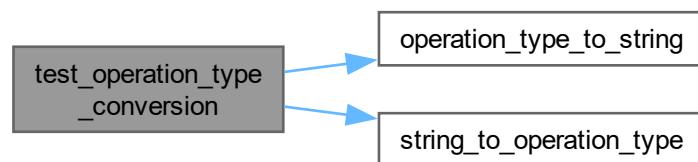


8.129.1.7 test_operation_type_conversion()

```
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()

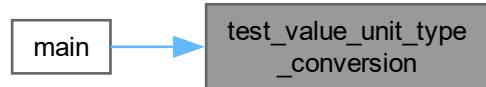
```
void test_value_unit_type_conversion (
    void ) [extern]
```

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:

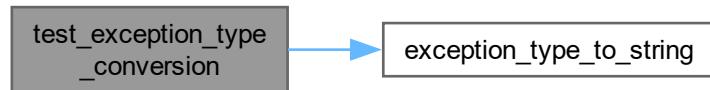


8.129.1.9 test_exception_type_conversion()

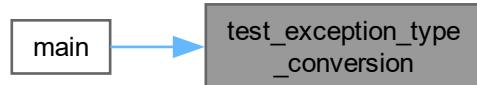
```
void test_exception_type_conversion (
    void ) [extern]
```

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:

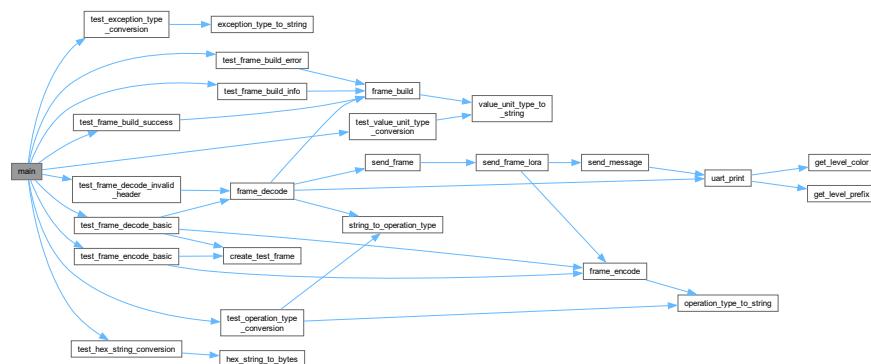


8.129.1.11 main()

```
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/stl.h"
00004 #include "hardware/uart.h"
00005
00006 // External test function declarations
00007 extern void test_frame_encode_basic(void);
00008 extern void test_frame_decode_basic(void);
00009 extern void test_frame_decode_invalid_header(void);
00010 extern void test_frame_build_success(void);
00011 extern void test_frame_build_error(void);
00012 extern void test_frame_build_info(void);
00013 extern void test_operation_type_conversion(void);
00014 extern void test_value_unit_type_conversion(void);
00015 extern void test_exception_type_conversion(void);
00016 extern void test_hex_string_conversion(void);
00017
00018 int main(void) {
00019     stdio_init_all();
00020     uart_init(uart0, 115200);
  
```

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

Index

_BH1750_DEFAULT_MTREG
 BH1750.h, [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_verbose, 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
 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
 SensorDataTypelIdentifier, 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
 MILLIAMP, 263
 NONE, 263, 264
 NOT_ALLOWED, 264
 operation_type_to_string, 264
 OperationType, 263
 PARAM_UNNECESSARY, 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, 69
 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
SensorDataTypelIdentifier
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