

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 handleTime() | 11 |
| 6.1.2.2 handleTimezoneOffset() | 12 |
| 6.1.2.3 handleClockSyncInterval() | 13 |
| 6.1.2.4 handleGetLastSyncTime() | 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 executeCommand() | 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 handleListCommands() | 17 |
| 6.3.2.2 handleGetBuildVersion() | 18 |
| 6.3.2.3 handleEnterBootloaderMode() | 18 |
| 6.4 Event Commands | 19 |
| 6.4.1 Detailed Description | 19 |
| 6.4.2 Function Documentation | 19 |
| 6.4.2.1 handleGetLastEvents() | 19 |
| 6.4.2.2 handleGetEventCount() | 20 |

| | |
|--|-----------|
| 6.5 GPS Commands | 21 |
| 6.5.1 Detailed Description | 21 |
| 6.5.2 Function Documentation | 21 |
| 6.5.2.1 handleGPSPowerStatus() | 21 |
| 6.5.2.2 handleEnableGPSTransparentMode() | 22 |
| 6.5.2.3 handleGetRMCDData() | 23 |
| 6.5.2.4 handleGetGGAData() | 24 |
| 6.6 Power Commands | 25 |
| 6.6.1 Detailed Description | 25 |
| 6.6.2 Function Documentation | 25 |
| 6.6.2.1 handleGetPowerManagerIDs() | 25 |
| 6.6.2.2 handleGetVoltageBattery() | 26 |
| 6.6.2.3 handleGetVoltage5V() | 27 |
| 6.6.2.4 handleGetCurrentChargeUSB() | 27 |
| 6.6.2.5 handleGetCurrentChargeSolar() | 28 |
| 6.6.2.6 handleGetCurrentChargeTotal() | 29 |
| 6.6.2.7 handleGetCurrentDraw() | 30 |
| 7 Class Documentation | 31 |
| 7.1 BH1750 Class Reference | 31 |
| 7.1.1 Detailed Description | 31 |
| 7.1.2 Member Enumeration Documentation | 31 |
| 7.1.2.1 Mode | 31 |
| 7.1.3 Constructor & Destructor Documentation | 32 |
| 7.1.3.1 BH1750() | 32 |
| 7.1.4 Member Function Documentation | 32 |
| 7.1.4.1 begin() | 32 |
| 7.1.4.2 configure() | 33 |
| 7.1.4.3 readLightLevel() | 33 |
| 7.1.4.4 write8() | 33 |
| 7.1.5 Member Data Documentation | 34 |
| 7.1.5.1 _i2c_addr | 34 |
| 7.2 BH1750Wrapper Class Reference | 34 |
| 7.2.1 Detailed Description | 35 |
| 7.2.2 Constructor & Destructor Documentation | 35 |
| 7.2.2.1 BH1750Wrapper() | 35 |
| 7.2.3 Member Function Documentation | 35 |
| 7.2.3.1 get_i2c_addr() | 35 |
| 7.2.3.2 init() | 35 |
| 7.2.3.3 readData() | 36 |
| 7.2.3.4 isInitialized() | 36 |
| 7.2.3.5 getType() | 36 |

| | |
|--|----|
| 7.2.3.6 configure() | 36 |
| 7.2.4 Member Data Documentation | 36 |
| 7.2.4.1 sensor | 36 |
| 7.2.4.2 initialized | 36 |
| 7.3 BME280 Class Reference | 37 |
| 7.3.1 Detailed Description | 38 |
| 7.3.2 Constructor & Destructor Documentation | 38 |
| 7.3.2.1 BME280() | 38 |
| 7.3.3 Member Function Documentation | 39 |
| 7.3.3.1 init() | 39 |
| 7.3.3.2 reset() | 39 |
| 7.3.3.3 read_raw_all() | 39 |
| 7.3.3.4 convert_temperature() | 39 |
| 7.3.3.5 convert_pressure() | 39 |
| 7.3.3.6 convert_humidity() | 40 |
| 7.3.3.7 configure_sensor() | 40 |
| 7.3.3.8 get_calibration_parameters() | 40 |
| 7.3.4 Member Data Documentation | 40 |
| 7.3.4.1 ADDR_SDO_LOW | 40 |
| 7.3.4.2 ADDR_SDO_HIGH | 41 |
| 7.3.4.3 i2c_port | 41 |
| 7.3.4.4 device_addr | 41 |
| 7.3.4.5 calib_params | 41 |
| 7.3.4.6 initialized | 41 |
| 7.3.4.7 t_fine | 41 |
| 7.3.4.8 REG_CONFIG | 41 |
| 7.3.4.9 REG_CTRL_MEAS | 41 |
| 7.3.4.10 REG_CTRL_HUM | 42 |
| 7.3.4.11 REG_RESET | 42 |
| 7.3.4.12 REG_PRESSURE_MSB | 42 |
| 7.3.4.13 REG_TEMPERATURE_MSB | 42 |
| 7.3.4.14 REG_HUMIDITY_MSB | 42 |
| 7.3.4.15 REG_DIG_T1_LSB | 42 |
| 7.3.4.16 REG_DIG_T1_MSB | 42 |
| 7.3.4.17 REG_DIG_T2_LSB | 42 |
| 7.3.4.18 REG_DIG_T2_MSB | 43 |
| 7.3.4.19 REG_DIG_T3_LSB | 43 |
| 7.3.4.20 REG_DIG_T3_MSB | 43 |
| 7.3.4.21 REG_DIG_P1_LSB | 43 |
| 7.3.4.22 REG_DIG_P1_MSB | 43 |
| 7.3.4.23 REG_DIG_P2_LSB | 43 |
| 7.3.4.24 REG_DIG_P2_MSB | 43 |

| | |
|---|----|
| 7.3.4.25 REG_DIG_P3_LSB | 43 |
| 7.3.4.26 REG_DIG_P3_MSB | 44 |
| 7.3.4.27 REG_DIG_P4_LSB | 44 |
| 7.3.4.28 REG_DIG_P4_MSB | 44 |
| 7.3.4.29 REG_DIG_P5_LSB | 44 |
| 7.3.4.30 REG_DIG_P5_MSB | 44 |
| 7.3.4.31 REG_DIG_P6_LSB | 44 |
| 7.3.4.32 REG_DIG_P6_MSB | 44 |
| 7.3.4.33 REG_DIG_P7_LSB | 44 |
| 7.3.4.34 REG_DIG_P7_MSB | 45 |
| 7.3.4.35 REG_DIG_P8_LSB | 45 |
| 7.3.4.36 REG_DIG_P8_MSB | 45 |
| 7.3.4.37 REG_DIG_P9_LSB | 45 |
| 7.3.4.38 REG_DIG_P9_MSB | 45 |
| 7.3.4.39 REG_DIG_H1 | 45 |
| 7.3.4.40 REG_DIG_H2 | 45 |
| 7.3.4.41 REG_DIG_H3 | 45 |
| 7.3.4.42 REG_DIG_H4 | 46 |
| 7.3.4.43 REG_DIG_H5 | 46 |
| 7.3.4.44 REG_DIG_H6 | 46 |
| 7.3.4.45 NUM_CALIB_PARAMS | 46 |
| 7.4 BME280CalibParam Struct Reference | 46 |
| 7.4.1 Detailed Description | 47 |
| 7.4.2 Member Data Documentation | 47 |
| 7.4.2.1 dig_t1 | 47 |
| 7.4.2.2 dig_t2 | 47 |
| 7.4.2.3 dig_t3 | 47 |
| 7.4.2.4 dig_p1 | 47 |
| 7.4.2.5 dig_p2 | 47 |
| 7.4.2.6 dig_p3 | 47 |
| 7.4.2.7 dig_p4 | 48 |
| 7.4.2.8 dig_p5 | 48 |
| 7.4.2.9 dig_p6 | 48 |
| 7.4.2.10 dig_p7 | 48 |
| 7.4.2.11 dig_p8 | 48 |
| 7.4.2.12 dig_p9 | 48 |
| 7.4.2.13 dig_h1 | 48 |
| 7.4.2.14 dig_h2 | 48 |
| 7.4.2.15 dig_h3 | 49 |
| 7.4.2.16 dig_h4 | 49 |
| 7.4.2.17 dig_h5 | 49 |
| 7.4.2.18 dig_h6 | 49 |

| | |
|--|----|
| 7.5 BME280Wrapper Class Reference | 49 |
| 7.5.1 Detailed Description | 50 |
| 7.5.2 Constructor & Destructor Documentation | 50 |
| 7.5.2.1 BME280Wrapper() | 50 |
| 7.5.3 Member Function Documentation | 51 |
| 7.5.3.1 init() | 51 |
| 7.5.3.2 readData() | 51 |
| 7.5.3.3 isInitialized() | 51 |
| 7.5.3.4 getType() | 51 |
| 7.5.3.5 configure() | 51 |
| 7.5.4 Member Data Documentation | 51 |
| 7.5.4.1 sensor | 51 |
| 7.5.4.2 initialized | 52 |
| 7.6 INA3221::conf_reg_t Struct Reference | 52 |
| 7.6.1 Detailed Description | 52 |
| 7.6.2 Member Data Documentation | 52 |
| 7.6.2.1 mode_shunt_en | 52 |
| 7.6.2.2 mode_bus_en | 52 |
| 7.6.2.3 mode_continious_en | 53 |
| 7.6.2.4 shunt_conv_time | 53 |
| 7.6.2.5 bus_conv_time | 53 |
| 7.6.2.6 avg_mode | 53 |
| 7.6.2.7 ch3_en | 53 |
| 7.6.2.8 ch2_en | 53 |
| 7.6.2.9 ch1_en | 53 |
| 7.6.2.10 reset | 54 |
| 7.7 DateTime Struct Reference | 54 |
| 7.7.1 Detailed Description | 54 |
| 7.7.2 Member Data Documentation | 54 |
| 7.7.2.1 year | 54 |
| 7.7.2.2 month | 54 |
| 7.7.2.3 day | 54 |
| 7.7.2.4 hour | 55 |
| 7.7.2.5 minute | 55 |
| 7.7.2.6 second | 55 |
| 7.7.2.7 weekday | 55 |
| 7.8 DS3231 Class Reference | 55 |
| 7.8.1 Detailed Description | 56 |
| 7.8.2 Constructor & Destructor Documentation | 56 |
| 7.8.2.1 DS3231() | 56 |
| 7.8.3 Member Function Documentation | 57 |
| 7.8.3.1 setTime() | 57 |

| | |
|--|----|
| 7.8.3.2 getTime() | 57 |
| 7.8.3.3 getTimeString() | 58 |
| 7.8.3.4 setTimeUnix() | 58 |
| 7.8.3.5 getTimeUnix() | 59 |
| 7.8.3.6 getDateTime() | 59 |
| 7.8.3.7 getTimeInteger() | 60 |
| 7.8.3.8 setTimezoneOffset() | 60 |
| 7.8.3.9 getTimezoneOffset() | 61 |
| 7.8.3.10 getTimeUnixLocal() | 61 |
| 7.8.3.11 getDateTimeLocal() | 61 |
| 7.8.3.12 setClockSyncInterval() | 61 |
| 7.8.3.13 getClockSyncInterval() | 62 |
| 7.8.3.14 setLastSyncTime() | 62 |
| 7.8.3.15 getLastSyncTime() | 62 |
| 7.8.3.16 bcd2bin() | 62 |
| 7.8.3.17 bin2bcd() | 62 |
| 7.8.3.18 preZero() | 63 |
| 7.8.3.19 dateTimeToUnix() | 63 |
| 7.8.3.20 unixToDateTIme() | 63 |
| 7.8.3.21 applyTimezone() | 64 |
| 7.8.4 Member Data Documentation | 64 |
| 7.8.4.1 WEEKDAYS | 64 |
| 7.8.4.2 i2c | 64 |
| 7.8.4.3 address | 65 |
| 7.8.4.4 RTC_REGISTER | 65 |
| 7.8.4.5 timezoneOffset | 65 |
| 7.8.4.6 syncInterval | 65 |
| 7.8.4.7 lastSyncTime | 65 |
| 7.9 EventEmitter Class Reference | 65 |
| 7.9.1 Detailed Description | 66 |
| 7.9.2 Member Function Documentation | 66 |
| 7.9.2.1 emit() | 66 |
| 7.10 EventLog Class Reference | 67 |
| 7.10.1 Detailed Description | 67 |
| 7.10.2 Member Function Documentation | 67 |
| 7.10.2.1 toString() | 67 |
| 7.10.3 Member Data Documentation | 68 |
| 7.10.3.1 id | 68 |
| 7.10.3.2 timestamp | 68 |
| 7.10.3.3 group | 68 |
| 7.10.3.4 event | 68 |
| 7.11 EventManager Class Reference | 69 |

| | |
|---|----|
| 7.11.1 Detailed Description | 70 |
| 7.11.2 Constructor & Destructor Documentation | 70 |
| 7.11.2.1 EventManager() | 70 |
| 7.11.2.2 ~EventManager() | 70 |
| 7.11.3 Member Function Documentation | 71 |
| 7.11.3.1 init() | 71 |
| 7.11.3.2 logEvent() | 71 |
| 7.11.3.3 getEvent() | 72 |
| 7.11.3.4 getEventCount() | 73 |
| 7.11.3.5 saveToStorage() | 73 |
| 7.11.3.6 loadFromStorage() | 73 |
| 7.11.4 Member Data Documentation | 74 |
| 7.11.4.1 events | 74 |
| 7.11.4.2 eventCount | 74 |
| 7.11.4.3 writeIndex | 74 |
| 7.11.4.4 eventMutex | 74 |
| 7.11.4.5 nextEventId | 74 |
| 7.11.4.6 needsPersistence | 74 |
| 7.12 EventManagerImpl Class Reference | 75 |
| 7.12.1 Detailed Description | 76 |
| 7.12.2 Constructor & Destructor Documentation | 76 |
| 7.12.2.1 EventManagerImpl() | 76 |
| 7.12.3 Member Function Documentation | 77 |
| 7.12.3.1 saveToStorage() | 77 |
| 7.12.3.2 loadFromStorage() | 77 |
| 7.13 FileHandle Struct Reference | 77 |
| 7.13.1 Detailed Description | 78 |
| 7.13.2 Member Data Documentation | 78 |
| 7.13.2.1 fd | 78 |
| 7.13.2.2 is_open | 78 |
| 7.14 Frame Struct Reference | 78 |
| 7.14.1 Detailed Description | 78 |
| 7.14.2 Member Data Documentation | 79 |
| 7.14.2.1 header | 79 |
| 7.14.2.2 direction | 79 |
| 7.14.2.3 operationType | 79 |
| 7.14.2.4 group | 79 |
| 7.14.2.5 command | 79 |
| 7.14.2.6 value | 79 |
| 7.14.2.7 unit | 79 |
| 7.14.2.8 footer | 80 |
| 7.15 HMC5883L Class Reference | 80 |

| | |
|---|----|
| 7.15.1 Detailed Description | 80 |
| 7.15.2 Constructor & Destructor Documentation | 80 |
| 7.15.2.1 HMC5883L() | 80 |
| 7.15.3 Member Function Documentation | 81 |
| 7.15.3.1 init() | 81 |
| 7.15.3.2 read() | 81 |
| 7.15.3.3 writeRegister() | 81 |
| 7.15.3.4 readRegisters() | 82 |
| 7.15.4 Member Data Documentation | 82 |
| 7.15.4.1 i2c | 82 |
| 7.15.4.2 address | 82 |
| 7.16 HMC5883LWrapper Class Reference | 83 |
| 7.16.1 Detailed Description | 84 |
| 7.16.2 Constructor & Destructor Documentation | 84 |
| 7.16.2.1 HMC5883LWrapper() | 84 |
| 7.16.3 Member Function Documentation | 84 |
| 7.16.3.1 init() | 84 |
| 7.16.3.2 readData() | 84 |
| 7.16.3.3 isInitialized() | 84 |
| 7.16.3.4 getType() | 85 |
| 7.16.3.5 configure() | 85 |
| 7.16.4 Member Data Documentation | 85 |
| 7.16.4.1 sensor | 85 |
| 7.16.4.2 initialized | 85 |
| 7.17 INA3221 Class Reference | 85 |
| 7.17.1 Detailed Description | 87 |
| 7.17.2 Constructor & Destructor Documentation | 87 |
| 7.17.2.1 INA3221() | 87 |
| 7.17.3 Member Function Documentation | 87 |
| 7.17.3.1 _read() | 87 |
| 7.17.3.2 _write() | 88 |
| 7.17.3.3 begin() | 89 |
| 7.17.3.4 setShuntRes() | 90 |
| 7.17.3.5 setFilterRes() | 90 |
| 7.17.3.6 setAddr() | 90 |
| 7.17.3.7 getReg() | 90 |
| 7.17.3.8 reset() | 91 |
| 7.17.3.9 setModePowerDown() | 91 |
| 7.17.3.10 setModeContinious() | 91 |
| 7.17.3.11 setModeTriggered() | 92 |
| 7.17.3.12 setShuntMeasEnable() | 92 |
| 7.17.3.13 setShuntMeasDisable() | 93 |

| | |
|--------------------------------------|-----|
| 7.17.3.14 setBusMeasEnable() | 93 |
| 7.17.3.15 setBusMeasDisable() | 93 |
| 7.17.3.16 setAveragingMode() | 94 |
| 7.17.3.17 setBusConversionTime() | 94 |
| 7.17.3.18 setShuntConversionTime() | 95 |
| 7.17.3.19 setCritAlertLimit() | 95 |
| 7.17.3.20 setWarnAlertLimit() | 95 |
| 7.17.3.21 setPwrValidUpLimit() | 96 |
| 7.17.3.22 setPwrValidLowLimit() | 96 |
| 7.17.3.23 setShuntSumAlertLimit() | 96 |
| 7.17.3.24 setCurrentSumAlertLimit() | 96 |
| 7.17.3.25 setWarnAlertLatchEnable() | 96 |
| 7.17.3.26 setWarnAlertLatchDisable() | 96 |
| 7.17.3.27 setCritAlertLatchEnable() | 96 |
| 7.17.3.28 setCritAlertLatchDisable() | 96 |
| 7.17.3.29 readFlags() | 96 |
| 7.17.3.30 getTimingCtrlAlertFlag() | 96 |
| 7.17.3.31 getPwrValidAlertFlag() | 97 |
| 7.17.3.32 getCurrentSumAlertFlag() | 97 |
| 7.17.3.33 getConversionReadyFlag() | 97 |
| 7.17.3.34 getManufID() | 97 |
| 7.17.3.35 getDieID() | 98 |
| 7.17.3.36 setChannelEnable() | 98 |
| 7.17.3.37 setChannelDisable() | 98 |
| 7.17.3.38 setWarnAlertShuntLimit() | 98 |
| 7.17.3.39 setCritAlertShuntLimit() | 98 |
| 7.17.3.40 setWarnAlertCurrentLimit() | 99 |
| 7.17.3.41 setCritAlertCurrentLimit() | 99 |
| 7.17.3.42 setCurrentSumEnable() | 99 |
| 7.17.3.43 setCurrentSumDisable() | 99 |
| 7.17.3.44 getShuntVoltage() | 99 |
| 7.17.3.45 getWarnAlertFlag() | 100 |
| 7.17.3.46 getCritAlertFlag() | 100 |
| 7.17.3.47 estimateOffsetVoltage() | 100 |
| 7.17.3.48 getCurrent() | 101 |
| 7.17.3.49 getCurrent_mA() | 101 |
| 7.17.3.50 getCurrentCompensated() | 101 |
| 7.17.3.51 getCurrentCompensated_mA() | 102 |
| 7.17.3.52 getVoltage() | 102 |
| 7.17.4 Member Data Documentation | 103 |
| 7.17.4.1 _i2c | 103 |
| 7.17.4.2 _i2c_addr | 103 |

| | |
|---|-----|
| 7.17.4.3 _shuntRes | 103 |
| 7.17.4.4 _filterRes | 103 |
| 7.17.4.5 _masken_reg | 103 |
| 7.18 ISensor Class Reference | 103 |
| 7.18.1 Detailed Description | 104 |
| 7.18.2 Constructor & Destructor Documentation | 104 |
| 7.18.2.1 ~ISensor() | 104 |
| 7.18.3 Member Function Documentation | 104 |
| 7.18.3.1 init() | 104 |
| 7.18.3.2 readData() | 104 |
| 7.18.3.3 isInitialized() | 104 |
| 7.18.3.4 getType() | 104 |
| 7.18.3.5 configure() | 105 |
| 7.19 LoRaClass Class Reference | 105 |
| 7.19.1 Detailed Description | 108 |
| 7.19.2 Constructor & Destructor Documentation | 108 |
| 7.19.2.1 LoRaClass() | 108 |
| 7.19.3 Member Function Documentation | 108 |
| 7.19.3.1 begin() | 108 |
| 7.19.3.2 end() | 109 |
| 7.19.3.3 beginPacket() | 109 |
| 7.19.3.4 endPacket() | 109 |
| 7.19.3.5 parsePacket() | 110 |
| 7.19.3.6 packetRssi() | 110 |
| 7.19.3.7 packetSnr() | 110 |
| 7.19.3.8 packetFrequencyError() | 111 |
| 7.19.3.9 rssi() | 111 |
| 7.19.3.10 write() [1/2] | 111 |
| 7.19.3.11 write() [2/2] | 112 |
| 7.19.3.12 available() | 112 |
| 7.19.3.13 read() | 113 |
| 7.19.3.14 peek() | 113 |
| 7.19.3.15 flush() | 114 |
| 7.19.3.16 onCadDone() | 114 |
| 7.19.3.17 onReceive() | 114 |
| 7.19.3.18 onTxDone() | 115 |
| 7.19.3.19 receive() | 115 |
| 7.19.3.20 channelActivityDetection() | 115 |
| 7.19.3.21 idle() | 116 |
| 7.19.3.22 sleep() | 116 |
| 7.19.3.23 setTxPower() | 117 |
| 7.19.3.24 setFrequency() | 117 |

| | |
|---|-----|
| 7.19.3.25 setSpreadingFactor() | 118 |
| 7.19.3.26 setSignalBandwidth() | 118 |
| 7.19.3.27 setCodingRate4() | 119 |
| 7.19.3.28 setPreambleLength() | 119 |
| 7.19.3.29 setSyncWord() | 119 |
| 7.19.3.30 enableCrc() | 120 |
| 7.19.3.31 disableCrc() | 120 |
| 7.19.3.32 enableInvertIQ() | 121 |
| 7.19.3.33 disableInvertIQ() | 121 |
| 7.19.3.34 setOCP() | 121 |
| 7.19.3.35 setGain() | 122 |
| 7.19.3.36 crc() | 122 |
| 7.19.3.37 noCrc() | 122 |
| 7.19.3.38 random() | 123 |
| 7.19.3.39 setPins() | 123 |
| 7.19.3.40 setSPI() | 123 |
| 7.19.3.41 setSPIFrequency() | 123 |
| 7.19.3.42 dumpRegisters() | 123 |
| 7.19.3.43 explicitHeaderMode() | 124 |
| 7.19.3.44 implicitHeaderMode() | 124 |
| 7.19.3.45 handleDio0Rise() | 125 |
| 7.19.3.46 isTransmitting() | 125 |
| 7.19.3.47 getSpreadingFactor() | 126 |
| 7.19.3.48 getSignalBandwidth() | 126 |
| 7.19.3.49 setLdoFlag() | 127 |
| 7.19.3.50 readRegister() | 127 |
| 7.19.3.51 writeRegister() | 129 |
| 7.19.3.52 singleTransfer() | 130 |
| 7.19.3.53 onDio0Rise() | 131 |
| 7.19.4 Member Data Documentation | 132 |
| 7.19.4.1 _spi | 132 |
| 7.19.4.2 _ss | 132 |
| 7.19.4.3 _reset | 132 |
| 7.19.4.4 _dio0 | 132 |
| 7.19.4.5 _frequency | 132 |
| 7.19.4.6 _packetIndex | 133 |
| 7.19.4.7 _implicitHeaderMode | 133 |
| 7.19.4.8 _onReceive | 133 |
| 7.19.4.9 _onCadDone | 133 |
| 7.19.4.10 _onTxDone | 133 |
| 7.20 INA3221::masken_reg_t Struct Reference | 133 |
| 7.20.1 Detailed Description | 134 |

| | |
|---|-----|
| 7.20.2 Member Data Documentation | 134 |
| 7.20.2.1 conv_ready | 134 |
| 7.20.2.2 timing_ctrl_alert | 134 |
| 7.20.2.3 pwr_valid_alert | 134 |
| 7.20.2.4 warn_alert_ch3 | 134 |
| 7.20.2.5 warn_alert_ch2 | 134 |
| 7.20.2.6 warn_alert_ch1 | 134 |
| 7.20.2.7 shunt_sum_alert | 135 |
| 7.20.2.8 crit_alert_ch3 | 135 |
| 7.20.2.9 crit_alert_ch2 | 135 |
| 7.20.2.10 crit_alert_ch1 | 135 |
| 7.20.2.11 crit_alert_latch_en | 135 |
| 7.20.2.12 warn_alert_latch_en | 135 |
| 7.20.2.13 shunt_sum_en_ch3 | 135 |
| 7.20.2.14 shunt_sum_en_ch2 | 135 |
| 7.20.2.15 shunt_sum_en_ch1 | 136 |
| 7.20.2.16 reserved | 136 |
| 7.21 MPU6050Wrapper Class Reference | 136 |
| 7.21.1 Detailed Description | 137 |
| 7.21.2 Constructor & Destructor Documentation | 137 |
| 7.21.2.1 MPU6050Wrapper() | 137 |
| 7.21.3 Member Function Documentation | 137 |
| 7.21.3.1 init() | 137 |
| 7.21.3.2 readData() | 137 |
| 7.21.3.3 isInitialized() | 138 |
| 7.21.3.4 getType() | 138 |
| 7.21.3.5 configure() | 138 |
| 7.21.4 Member Data Documentation | 138 |
| 7.21.4.1 sensor | 138 |
| 7.21.4.2 initialized | 138 |
| 7.22 NMEAData Class Reference | 138 |
| 7.22.1 Detailed Description | 139 |
| 7.22.2 Constructor & Destructor Documentation | 139 |
| 7.22.2.1 NMEAData() | 139 |
| 7.22.3 Member Function Documentation | 139 |
| 7.22.3.1 updateRmcTokens() | 139 |
| 7.22.3.2 updateGgaTokens() | 139 |
| 7.22.3.3 getRmcTokens() | 139 |
| 7.22.3.4 getGgaTokens() | 139 |
| 7.22.4 Member Data Documentation | 140 |
| 7.22.4.1 rmcTokens | 140 |
| 7.22.4.2 ggaTokens | 140 |

| | |
|---|-----|
| 7.22.4.3 rmc_mutex | 140 |
| 7.22.4.4 gga_mutex | 140 |
| 7.23 PowerManager Class Reference | 140 |
| 7.23.1 Detailed Description | 141 |
| 7.23.2 Constructor & Destructor Documentation | 141 |
| 7.23.2.1 PowerManager() | 141 |
| 7.23.3 Member Function Documentation | 141 |
| 7.23.3.1 initialize() | 141 |
| 7.23.3.2 readIDs() | 142 |
| 7.23.3.3 getCurrentChargeSolar() | 142 |
| 7.23.3.4 getCurrentChargeUSB() | 142 |
| 7.23.3.5 getCurrentChargeTotal() | 142 |
| 7.23.3.6 getCurrentDraw() | 142 |
| 7.23.3.7 getVoltageBattery() | 143 |
| 7.23.3.8 getVoltage5V() | 143 |
| 7.23.3.9 configure() | 143 |
| 7.23.3.10 isSolarActive() | 143 |
| 7.23.3.11 isUSBConnected() | 144 |
| 7.23.4 Member Data Documentation | 144 |
| 7.23.4.1 SOLAR_CURRENT_THRESHOLD | 144 |
| 7.23.4.2 USB_CURRENT_THRESHOLD | 144 |
| 7.23.4.3 VOLTAGE_LOW_THRESHOLD | 144 |
| 7.23.4.4 VOLTAGE_OVERCHARGE_THRESHOLD | 145 |
| 7.23.4.5 FALL_RATE_THRESHOLD | 145 |
| 7.23.4.6 FALLING_TREND_REQUIRED | 145 |
| 7.23.4.7 ina3221 | 145 |
| 7.23.4.8 initialized | 145 |
| 7.23.4.9 solarActive | 145 |
| 7.23.4.10 usbConnected | 145 |
| 7.24 Print Class Reference | 146 |
| 7.24.1 Detailed Description | 147 |
| 7.24.2 Constructor & Destructor Documentation | 147 |
| 7.24.2.1 Print() | 147 |
| 7.24.3 Member Function Documentation | 147 |
| 7.24.3.1 printNumber() | 147 |
| 7.24.3.2 printULLNumber() | 148 |
| 7.24.3.3 printFloat() | 149 |
| 7.24.3.4 setWriteError() | 149 |
| 7.24.3.5 getWriteError() | 150 |
| 7.24.3.6 clearWriteError() | 150 |
| 7.24.3.7 write() [1/4] | 150 |
| 7.24.3.8 write() [2/4] | 151 |

| | |
|---|-----|
| 7.24.3.9 write() [3/4] | 152 |
| 7.24.3.10 write() [4/4] | 152 |
| 7.24.3.11 availableForWrite() | 153 |
| 7.24.3.12 print() [1/11] | 153 |
| 7.24.3.13 print() [2/11] | 154 |
| 7.24.3.14 print() [3/11] | 155 |
| 7.24.3.15 print() [4/11] | 155 |
| 7.24.3.16 print() [5/11] | 156 |
| 7.24.3.17 print() [6/11] | 156 |
| 7.24.3.18 print() [7/11] | 157 |
| 7.24.3.19 print() [8/11] | 157 |
| 7.24.3.20 print() [9/11] | 158 |
| 7.24.3.21 print() [10/11] | 158 |
| 7.24.3.22 print() [11/11] | 159 |
| 7.24.3.23 println() [1/11] | 159 |
| 7.24.3.24 println() [2/11] | 159 |
| 7.24.3.25 println() [3/11] | 160 |
| 7.24.3.26 println() [4/11] | 160 |
| 7.24.3.27 println() [5/11] | 161 |
| 7.24.3.28 println() [6/11] | 161 |
| 7.24.3.29 println() [7/11] | 162 |
| 7.24.3.30 println() [8/11] | 162 |
| 7.24.3.31 println() [9/11] | 163 |
| 7.24.3.32 println() [10/11] | 163 |
| 7.24.3.33 println() [11/11] | 164 |
| 7.24.3.34 flush() | 165 |
| 7.24.4 Member Data Documentation | 165 |
| 7.24.4.1 write_error | 165 |
| 7.25 SensorWrapper Class Reference | 166 |
| 7.25.1 Detailed Description | 166 |
| 7.25.2 Constructor & Destructor Documentation | 167 |
| 7.25.2.1 SensorWrapper() | 167 |
| 7.25.3 Member Function Documentation | 167 |
| 7.25.3.1 getInstance() | 167 |
| 7.25.3.2 initSensor() | 167 |
| 7.25.3.3 configureSensor() | 168 |
| 7.25.3.4 readSensorData() | 168 |
| 7.25.4 Member Data Documentation | 169 |
| 7.25.4.1 sensors | 169 |
| 8 File Documentation | 171 |
| 8.1 build_number.h File Reference | 171 |

| | |
|--|-----|
| 8.1.1 Macro Definition Documentation | 171 |
| 8.1.1.1 BUILD_NUMBER | 171 |
| 8.2 build_number.h | 171 |
| 8.3 includes.h File Reference | 172 |
| 8.4 includes.h | 173 |
| 8.5 lib/clock/DS3231.cpp File Reference | 173 |
| 8.6 DS3231.cpp | 174 |
| 8.7 lib/clock/DS3231.h File Reference | 176 |
| 8.8 DS3231.h | 177 |
| 8.9 lib/comms/commands/clock_commands.cpp File Reference | 177 |
| 8.10 clock_commands.cpp | 178 |
| 8.11 lib/comms/commands/commands.cpp File Reference | 180 |
| 8.12 commands.cpp | 181 |
| 8.13 lib/comms/commands/commands.h File Reference | 181 |
| 8.13.1 Function Documentation | 183 |
| 8.13.1.1 handleGetCommandsTimestamp() | 183 |
| 8.14 commands.h | 183 |
| 8.15 lib/comms/commands/diagnostic_commands.cpp File Reference | 184 |
| 8.16 diagnostic_commands.cpp | 184 |
| 8.17 lib/comms/commands/event_commands.cpp File Reference | 185 |
| 8.18 event_commands.cpp | 186 |
| 8.19 lib/comms/commands/gps_commands.cpp File Reference | 187 |
| 8.20 gps_commands.cpp | 187 |
| 8.21 lib/comms/commands/power_commands.cpp File Reference | 190 |
| 8.22 power_commands.cpp | 190 |
| 8.23 lib/comms/communication.cpp File Reference | 192 |
| 8.23.1 Function Documentation | 192 |
| 8.23.1.1 initializeRadio() | 192 |
| 8.23.2 Variable Documentation | 193 |
| 8.23.2.1 outgoing | 193 |
| 8.23.2.2 msgCount | 193 |
| 8.23.2.3 lastSendTime | 194 |
| 8.23.2.4 lastReceiveTime | 194 |
| 8.23.2.5 lastPrintTime | 194 |
| 8.23.2.6 interval | 194 |
| 8.24 communication.cpp | 194 |
| 8.25 lib/comms/communication.h File Reference | 195 |
| 8.25.1 Function Documentation | 196 |
| 8.25.1.1 initializeRadio() | 196 |
| 8.25.1.2 sendMessage() | 197 |
| 8.25.1.3 sendLargePacket() | 197 |
| 8.25.1.4 onReceive() | 197 |

| | |
|--|-----|
| 8.25.1.5 handleUartInput() | 198 |
| 8.25.1.6 processFrameData() | 199 |
| 8.25.1.7 handleCommandFrame() | 200 |
| 8.25.1.8 sendEventRegister() | 200 |
| 8.25.1.9 encodeFrame() | 200 |
| 8.25.1.10 decodeFrame() | 201 |
| 8.25.1.11 buildFrame() | 202 |
| 8.25.1.12 determineUnit() | 204 |
| 8.25.1.13 sendFrame() | 205 |
| 8.26 communication.h | 205 |
| 8.27 lib/comms/frame.cpp File Reference | 206 |
| 8.27.1 Detailed Description | 207 |
| 8.27.2 Typedef Documentation | 207 |
| 8.27.2.1 CommandHandler | 207 |
| 8.27.3 Function Documentation | 207 |
| 8.27.3.1 encodeFrame() | 207 |
| 8.27.3.2 decodeFrame() | 208 |
| 8.27.3.3 processFrameData() | 209 |
| 8.27.3.4 sendEventRegister() | 210 |
| 8.27.3.5 buildFrame() | 210 |
| 8.27.4 Variable Documentation | 212 |
| 8.27.4.1 eventRegister | 212 |
| 8.28 frame.cpp | 213 |
| 8.29 lib/comms/LoRa/LoRa-RP2040.cpp File Reference | 214 |
| 8.29.1 Macro Definition Documentation | 216 |
| 8.29.1.1 REG_FIFO | 216 |
| 8.29.1.2 REG_OP_MODE | 216 |
| 8.29.1.3 REG_FRF_MSB | 216 |
| 8.29.1.4 REG_FRF_MID | 216 |
| 8.29.1.5 REG_FRF_LSB | 216 |
| 8.29.1.6 REG_PA_CONFIG | 216 |
| 8.29.1.7 IRQ_CAD_DETECTED_MASK | 216 |
| 8.29.1.8 REG_OCP | 217 |
| 8.29.1.9 REG_LNA | 217 |
| 8.29.1.10 IRQ_CAD_DONE_MASK | 217 |
| 8.29.1.11 REG_FIFO_ADDR_PTR | 217 |
| 8.29.1.12 REG_FIFO_TX_BASE_ADDR | 217 |
| 8.29.1.13 REG_FIFO_RX_BASE_ADDR | 217 |
| 8.29.1.14 REG_FIFO_RX_CURRENT_ADDR | 217 |
| 8.29.1.15 REG_IRQ_FLAGS | 217 |
| 8.29.1.16 REG_RX_NB_BYTES | 218 |
| 8.29.1.17 REG_PKT_SNR_VALUE | 218 |

| | |
|--|-----|
| 8.29.1.18 REG_PKT_RSSI_VALUE | 218 |
| 8.29.1.19 REG_RSSI_VALUE | 218 |
| 8.29.1.20 REG_MODEM_CONFIG_1 | 218 |
| 8.29.1.21 REG_MODEM_CONFIG_2 | 218 |
| 8.29.1.22 REG_PREAMBLE_MSB | 218 |
| 8.29.1.23 REG_PREAMBLE_LSB | 218 |
| 8.29.1.24 REG_PAYLOAD_LENGTH | 219 |
| 8.29.1.25 REG_MODEM_CONFIG_3 | 219 |
| 8.29.1.26 REG_FREQ_ERROR_MSB | 219 |
| 8.29.1.27 REG_FREQ_ERROR_MID | 219 |
| 8.29.1.28 REG_FREQ_ERROR_LSB | 219 |
| 8.29.1.29 REG_RSSI_WIDEBAND | 219 |
| 8.29.1.30 REG_DETECTION_OPTIMIZE | 219 |
| 8.29.1.31 REG_INVERTIQ | 219 |
| 8.29.1.32 REG_DETECTION_THRESHOLD | 220 |
| 8.29.1.33 REG_SYNC_WORD | 220 |
| 8.29.1.34 REG_INVERTIQ2 | 220 |
| 8.29.1.35 REG_DIO_MAPPING_1 | 220 |
| 8.29.1.36 REG_VERSION | 220 |
| 8.29.1.37 REG_PA_DAC | 220 |
| 8.29.1.38 MODE_CAD | 220 |
| 8.29.1.39 MODE_LONG_RANGE_MODE | 220 |
| 8.29.1.40 MODE_SLEEP | 221 |
| 8.29.1.41 MODE_STDBY | 221 |
| 8.29.1.42 MODE_TX | 221 |
| 8.29.1.43 MODE_RX_CONTINUOUS | 221 |
| 8.29.1.44 MODE_RX_SINGLE | 221 |
| 8.29.1.45 PA_BOOST | 221 |
| 8.29.1.46 IRQ_TX_DONE_MASK | 221 |
| 8.29.1.47 IRQ_PAYLOAD_CRC_ERROR_MASK | 221 |
| 8.29.1.48 IRQ_RX_DONE_MASK | 222 |
| 8.29.1.49 RF_MID_BAND_THRESHOLD | 222 |
| 8.29.1.50 RSSI_OFFSET_HF_PORT | 222 |
| 8.29.1.51 RSSI_OFFSET_LF_PORT | 222 |
| 8.29.1.52 MAX_PKT_LENGTH | 222 |
| 8.29.1.53 ISR_PREFIX | 222 |
| 8.29.2 Variable Documentation | 222 |
| 8.29.2.1 LoRa | 222 |
| 8.30 LoRa-RP2040.cpp | 223 |
| 8.31 lib/comms/LoRa/LoRa-RP2040.h File Reference | 231 |
| 8.31.1 Function Documentation | 232 |
| 8.31.1.1 __empty() | 232 |

| | |
|--|-----|
| 8.31.2 Variable Documentation | 232 |
| 8.31.2.1 LoRa | 232 |
| 8.32 LoRa-RP2040.h | 233 |
| 8.33 lib/comms/LoRa/Print.cpp File Reference | 234 |
| 8.34 Print.cpp | 234 |
| 8.35 lib/comms/LoRa/Print.h File Reference | 239 |
| 8.35.1 Macro Definition Documentation | 239 |
| 8.35.1.1 DEC | 239 |
| 8.35.1.2 HEX | 240 |
| 8.35.1.3 OCT | 240 |
| 8.35.1.4 BIN | 240 |
| 8.36 Print.h | 240 |
| 8.37 lib/comms/protocol.h File Reference | 241 |
| 8.37.1 Enumeration Type Documentation | 242 |
| 8.37.1.1 ExecutionResult | 242 |
| 8.37.1.2 OperationType | 243 |
| 8.37.1.3 CommandAccessLevel | 243 |
| 8.37.1.4 ValueUnit | 243 |
| 8.37.1.5 ExceptionType | 243 |
| 8.37.2 Function Documentation | 244 |
| 8.37.2.1 exceptionTypeToString() | 244 |
| 8.37.2.2 operationTypeToString() | 244 |
| 8.37.2.3 stringToOperationType() | 245 |
| 8.37.2.4 hexStringToBytes() | 245 |
| 8.37.2.5 valueUnitTypeToString() | 245 |
| 8.37.3 Variable Documentation | 246 |
| 8.37.3.1 FRAME_BEGIN | 246 |
| 8.37.3.2 FRAME_END | 247 |
| 8.37.3.3 DELIMITER | 247 |
| 8.38 protocol.h | 247 |
| 8.39 lib/comms/receive.cpp File Reference | 248 |
| 8.39.1 Detailed Description | 248 |
| 8.39.2 Function Documentation | 248 |
| 8.39.2.1 onReceive() | 248 |
| 8.39.2.2 handleUartInput() | 249 |
| 8.40 receive.cpp | 250 |
| 8.41 lib/comms/send.cpp File Reference | 251 |
| 8.41.1 Detailed Description | 251 |
| 8.41.2 Function Documentation | 251 |
| 8.41.2.1 sendMessage() | 251 |
| 8.41.2.2 sendFrame() | 252 |
| 8.41.2.3 sendLargePacket() | 253 |

| | |
|--|-----|
| 8.42 send.cpp | 254 |
| 8.43 lib/comms/utils_converters.cpp File Reference | 254 |
| 8.43.1 Detailed Description | 255 |
| 8.43.2 Function Documentation | 255 |
| 8.43.2.1 exceptionTypeToString() | 255 |
| 8.43.2.2 valueUnitTypeToString() | 256 |
| 8.43.2.3 operationTypeToString() | 257 |
| 8.43.2.4 stringToOperationType() | 257 |
| 8.43.2.5 hexStringToBytes() | 258 |
| 8.44 utils_converters.cpp | 258 |
| 8.45 lib/eventman/event_manager.cpp File Reference | 259 |
| 8.45.1 Detailed Description | 260 |
| 8.45.2 Function Documentation | 260 |
| 8.45.2.1 checkPowerEvents() | 260 |
| 8.45.3 Variable Documentation | 261 |
| 8.45.3.1 eventLogId | 261 |
| 8.45.3.2 lastPowerState | 261 |
| 8.45.3.3 FALL_RATE_THRESHOLD | 261 |
| 8.45.3.4 FALLING_TREND_REQUIRED | 261 |
| 8.45.3.5 VOLTAGE_LOW_THRESHOLD | 262 |
| 8.45.3.6 VOLTAGE_OVERCHARGE_THRESHOLD | 262 |
| 8.45.3.7 fallingTrendCount | 262 |
| 8.45.3.8 lastSolarState | 262 |
| 8.45.3.9 lastUSBState | 262 |
| 8.45.3.10 systemClock | 262 |
| 8.45.3.11 eventManager | 263 |
| 8.46 event_manager.cpp | 263 |
| 8.47 lib/eventman/event_manager.h File Reference | 265 |
| 8.47.1 Macro Definition Documentation | 266 |
| 8.47.1.1 EVENT_BUFFER_SIZE | 266 |
| 8.47.2 Enumeration Type Documentation | 266 |
| 8.47.2.1 EventGroup | 266 |
| 8.47.2.2 SystemEvent | 267 |
| 8.47.2.3 PowerEvent | 267 |
| 8.47.2.4 CommsEvent | 267 |
| 8.47.2.5 GPSEvent | 268 |
| 8.47.2.6 ClockEvent | 268 |
| 8.47.3 Function Documentation | 268 |
| 8.47.3.1 __attribute__(). | 268 |
| 8.47.3.2 toString() | 269 |
| 8.47.3.3 checkPowerEvents() | 269 |
| 8.47.4 Variable Documentation | 270 |

| | |
|---|-----|
| 8.47.4.1 id | 270 |
| 8.47.4.2 timestamp | 270 |
| 8.47.4.3 group | 270 |
| 8.47.4.4 event | 270 |
| 8.47.4.5 __attribute__ | 270 |
| 8.47.4.6 eventManager | 270 |
| 8.48 event_manager.h | 271 |
| 8.49 lib/location/gps_collector.cpp File Reference | 272 |
| 8.49.1 Macro Definition Documentation | 273 |
| 8.49.1.1 MAX_RAW_DATA_LENGTH | 273 |
| 8.49.2 Function Documentation | 274 |
| 8.49.2.1 splitString() | 274 |
| 8.49.2.2 collectGPSData() | 274 |
| 8.49.3 Variable Documentation | 275 |
| 8.49.3.1 nmea_data | 275 |
| 8.50 gps_collector.cpp | 275 |
| 8.51 lib/location/gps_collector.h File Reference | 276 |
| 8.51.1 Function Documentation | 276 |
| 8.51.1.1 collectGPSData() | 276 |
| 8.52 gps_collector.h | 277 |
| 8.53 lib/location/NMEA/NMEA_data.cpp File Reference | 278 |
| 8.53.1 Variable Documentation | 278 |
| 8.53.1.1 nmea_data | 278 |
| 8.54 NMEA_data.cpp | 278 |
| 8.55 lib/location/NMEA/NMEA_data.h File Reference | 279 |
| 8.55.1 Variable Documentation | 280 |
| 8.55.1.1 nmea_data | 280 |
| 8.56 NMEA_data.h | 280 |
| 8.57 lib/pin_config.cpp File Reference | 281 |
| 8.57.1 Variable Documentation | 281 |
| 8.57.1.1 csPin | 281 |
| 8.57.1.2 resetPin | 282 |
| 8.57.1.3 irqPin | 282 |
| 8.57.1.4 localAddress | 282 |
| 8.57.1.5 destination | 282 |
| 8.58 pin_config.cpp | 282 |
| 8.59 lib/pin_config.h File Reference | 283 |
| 8.59.1 Macro Definition Documentation | 284 |
| 8.59.1.1 DEBUG_UART_PORT | 284 |
| 8.59.1.2 DEBUG_UART_BAUD_RATE | 284 |
| 8.59.1.3 DEBUG_UART_TX_PIN | 284 |
| 8.59.1.4 DEBUG_UART_RX_PIN | 284 |

| | |
|--|-----|
| 8.59.1.5 MAIN_I2C_PORT | 285 |
| 8.59.1.6 MAIN_I2C_SDA_PIN | 285 |
| 8.59.1.7 MAIN_I2C_SCL_PIN | 285 |
| 8.59.1.8 GPS_UART_PORT | 285 |
| 8.59.1.9 GPS_UART_BAUD_RATE | 285 |
| 8.59.1.10 GPS_UART_TX_PIN | 285 |
| 8.59.1.11 GPS_UART_RX_PIN | 285 |
| 8.59.1.12 GPS_POWER_ENABLE_PIN | 285 |
| 8.59.1.13 BUFFER_SIZE | 286 |
| 8.59.1.14 SD_SPI_PORT | 286 |
| 8.59.1.15 SD_MISO_PIN | 286 |
| 8.59.1.16 SD_MOSI_PIN | 286 |
| 8.59.1.17 SD_SCK_PIN | 286 |
| 8.59.1.18 SD_CS_PIN | 286 |
| 8.59.1.19 SD_CARD_DETECT_PIN | 286 |
| 8.59.1.20 SX1278_MISO | 286 |
| 8.59.1.21 SX1278_CS | 287 |
| 8.59.1.22 SX1278_SCK | 287 |
| 8.59.1.23 SX1278_MOSI | 287 |
| 8.59.1.24 SPI_PORT | 287 |
| 8.59.1.25 READ_BIT | 287 |
| 8.59.1.26 LORA_DEFAULT_SPI | 287 |
| 8.59.1.27 LORA_DEFAULT_SPI_FREQUENCY | 287 |
| 8.59.1.28 LORA_DEFAULT_SS_PIN | 287 |
| 8.59.1.29 LORA_DEFAULT_RESET_PIN | 288 |
| 8.59.1.30 LORA_DEFAULT_DIO0_PIN | 288 |
| 8.59.1.31 PA_OUTPUT_RFO_PIN | 288 |
| 8.59.1.32 PA_OUTPUT_PA_BOOST_PIN | 288 |
| 8.59.2 Variable Documentation | 288 |
| 8.59.2.1 csPin | 288 |
| 8.59.2.2 resetPin | 288 |
| 8.59.2.3 irqPin | 288 |
| 8.59.2.4 localAddress | 289 |
| 8.59.2.5 destination | 289 |
| 8.60 pin_config.h | 289 |
| 8.61 lib/powerman/INA3221/INA3221.cpp File Reference | 290 |
| 8.62 INA3221.cpp | 290 |
| 8.63 lib/powerman/INA3221/INA3221.h File Reference | 294 |
| 8.63.1 Enumeration Type Documentation | 295 |
| 8.63.1.1 ina3221_addr_t | 295 |
| 8.63.1.2 ina3221_ch_t | 295 |
| 8.63.1.3 ina3221_reg_t | 295 |

| | |
|--|-----|
| 8.63.1.4 <code>ina3221_conv_time_t</code> | 296 |
| 8.63.1.5 <code>ina3221_avg_mode_t</code> | 296 |
| 8.63.2 Variable Documentation | 297 |
| 8.63.2.1 <code>INA3221_CH_NUM</code> | 297 |
| 8.63.2.2 <code>SHUNT_VOLTAGE_LSB_UV</code> | 297 |
| 8.64 <code>INA3221.h</code> | 297 |
| 8.65 <code>lib/powerman/PowerManager.cpp</code> File Reference | 301 |
| 8.66 <code>PowerManager.cpp</code> | 301 |
| 8.67 <code>lib/powerman/PowerManager.h</code> File Reference | 302 |
| 8.68 <code>PowerManager.h</code> | 303 |
| 8.69 <code>lib/sensors/BH1750/BH1750.cpp</code> File Reference | 304 |
| 8.70 <code>BH1750.cpp</code> | 304 |
| 8.71 <code>lib/sensors/BH1750/BH1750.h</code> File Reference | 305 |
| 8.71.1 Macro Definition Documentation | 306 |
| 8.71.1.1 <code>_BH1750_DEVICE_ID</code> | 306 |
| 8.71.1.2 <code>_BH1750_MTREG_MIN</code> | 306 |
| 8.71.1.3 <code>_BH1750_MTREG_MAX</code> | 307 |
| 8.71.1.4 <code>_BH1750_DEFAULT_MTREG</code> | 307 |
| 8.72 <code>BH1750.h</code> | 307 |
| 8.73 <code>lib/sensors/BH1750/BH1750_WRAPPER.cpp</code> File Reference | 307 |
| 8.74 <code>BH1750_WRAPPER.cpp</code> | 308 |
| 8.75 <code>lib/sensors/BH1750/BH1750_WRAPPER.h</code> File Reference | 309 |
| 8.76 <code>BH1750_WRAPPER.h</code> | 310 |
| 8.77 <code>lib/sensors/BME280/BME280.cpp</code> File Reference | 310 |
| 8.78 <code>BME280.cpp</code> | 311 |
| 8.79 <code>lib/sensors/BME280/BME280.h</code> File Reference | 314 |
| 8.80 <code>BME280.h</code> | 315 |
| 8.81 <code>lib/sensors/BME280/BME280_WRAPPER.cpp</code> File Reference | 317 |
| 8.82 <code>BME280_WRAPPER.cpp</code> | 317 |
| 8.83 <code>lib/sensors/BME280/BME280_WRAPPER.h</code> File Reference | 318 |
| 8.84 <code>BME280_WRAPPER.h</code> | 318 |
| 8.85 <code>lib/sensors/HMC5883L/HMC5883L.cpp</code> File Reference | 319 |
| 8.86 <code>HMC5883L.cpp</code> | 319 |
| 8.87 <code>lib/sensors/HMC5883L/HMC5883L.h</code> File Reference | 320 |
| 8.88 <code>HMC5883L.h</code> | 321 |
| 8.89 <code>lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp</code> File Reference | 321 |
| 8.90 <code>HMC5883L_WRAPPER.cpp</code> | 322 |
| 8.91 <code>lib/sensors/HMC5883L/HMC5883L_WRAPPER.h</code> File Reference | 323 |
| 8.92 <code>HMC5883L_WRAPPER.h</code> | 324 |
| 8.93 <code>lib/sensors/ISensor.cpp</code> File Reference | 324 |
| 8.93.1 Detailed Description | 325 |
| 8.94 <code>ISensor.cpp</code> | 325 |

| | |
|--|-----|
| 8.95 lib/sensors/ISensor.h File Reference | 325 |
| 8.95.1 Enumeration Type Documentation | 326 |
| 8.95.1.1 SensorType | 326 |
| 8.95.1.2 SensorDataTypIdentifier | 327 |
| 8.96 ISensor.h | 327 |
| 8.97 lib/sensors/MPU6050/MPU6050.cpp File Reference | 328 |
| 8.98 MPU6050.cpp | 328 |
| 8.99 lib/sensors/MPU6050/MPU6050.h File Reference | 329 |
| 8.100 MPU6050.h | 329 |
| 8.101 lib/sensors/MPU6050/MPU6050_WRAPPER.cpp File Reference | 329 |
| 8.102 MPU6050_WRAPPER.cpp | 329 |
| 8.103 lib/sensors/MPU6050/MPU6050_WRAPPER.h File Reference | 329 |
| 8.104 MPU6050_WRAPPER.h | 330 |
| 8.105 lib/storage/storage.cpp File Reference | 331 |
| 8.105.1 Detailed Description | 332 |
| 8.105.2 Function Documentation | 332 |
| 8.105.2.1 fs_init() | 332 |
| 8.105.2.2 fs_open_file() | 332 |
| 8.105.2.3 fs_write_file() | 333 |
| 8.105.2.4 fs_read_file() | 333 |
| 8.105.2.5 fs_close_file() | 333 |
| 8.105.2.6 fs_file_exists() | 334 |
| 8.106 storage.cpp | 334 |
| 8.107 lib/storage/storage.h File Reference | 336 |
| 8.107.1 Function Documentation | 337 |
| 8.107.1.1 fs_init() | 337 |
| 8.107.1.2 fs_open_file() | 337 |
| 8.107.1.3 fs_write_file() | 338 |
| 8.107.1.4 fs_read_file() | 338 |
| 8.107.1.5 fs_close_file() | 338 |
| 8.107.1.6 fs_file_exists() | 339 |
| 8.108 storage.h | 339 |
| 8.109 lib/utils.cpp File Reference | 340 |
| 8.109.1 Detailed Description | 341 |
| 8.109.2 Macro Definition Documentation | 341 |
| 8.109.2.1 LOG_FILENAME | 341 |
| 8.109.3 Function Documentation | 341 |
| 8.109.3.1 uartPrint() | 341 |
| 8.109.3.2 crc16() | 342 |
| 8.109.4 Variable Documentation | 343 |
| 8.109.4.1 uart_mutex | 343 |
| 8.110 utils.cpp | 343 |

| | |
|--|-----|
| 8.111 lib/utils.h File Reference | 343 |
| 8.111.1 Function Documentation | 344 |
| 8.111.1.1 uartPrint() | 344 |
| 8.111.1.2 crc16() | 345 |
| 8.112 utils.h | 346 |
| 8.113 main.cpp File Reference | 346 |
| 8.113.1 Macro Definition Documentation | 346 |
| 8.113.1.1 LOG_FILENAME | 346 |
| 8.113.2 Function Documentation | 347 |
| 8.113.2.1 core1_entry() | 347 |
| 8.113.2.2 initSystems() | 347 |
| 8.113.2.3 loggingRoutine() | 348 |
| 8.113.2.4 main() | 348 |
| 8.113.3 Variable Documentation | 348 |
| 8.113.3.1 powerManager | 348 |
| 8.113.3.2 systemClock | 349 |
| 8.113.3.3 buffer | 349 |
| 8.113.3.4 bufferIndex | 349 |
| 8.114 main.cpp | 349 |

Chapter 1

Clock Commands

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

Command ID: 3.3

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

Command ID: 7.2

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

Command ID: 2

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

Command ID: 1

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

Command ID: 2.5

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

Command ID: 2.6

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

Command ID: 2.4

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

Command ID: 2.7

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

Command ID: 5.2

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

Command ID: 7.4

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

Command ID: 5.1

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

Command ID: 3.7

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

Command ID: 2.0

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

Command ID: 7.3

Member `handleGetVoltage5V (const std::string ¶m, OperationType operationType)`

Command ID: 2.3

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

Command ID: 2.2

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

Command ID: 7.1

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

Command ID: 0

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

Command ID: 3.0

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

Command ID: 3.1

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 | 19 |
| GPS Commands | 21 |
| Power Commands | 25 |

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

| | |
|---------------------------------|-----|
| BH1750 | 31 |
| BME280 | 37 |
| BME280CalibParam | 46 |
| INA3221::conf_reg_t | 52 |
| DateTime | 54 |
| DS3231 | 55 |
| EventEmitter | 65 |
| EventLog | 67 |
| EventManager | 69 |
| EventManagerImpl | 75 |
| FileHandle | 77 |
| Frame | 78 |
| HMC5883L | 80 |
| INA3221 | 85 |
| ISensor | 103 |
| BH1750Wrapper | 34 |
| BME280Wrapper | 49 |
| HMC5883LWrapper | 83 |
| MPU6050Wrapper | 136 |
| INA3221::masken_reg_t | 133 |
| NMEAData | 138 |
| PowerManager | 140 |
| Print | 146 |
| LoRaClass | 105 |
| SensorWrapper | 166 |

Chapter 4

Class Index

4.1 Class List

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

| | |
|---|-----|
| BH1750 | 31 |
| BH1750Wrapper | 34 |
| BME280 | 37 |
| BME280CalibParam | 46 |
| BME280Wrapper | 49 |
| INA3221::conf_reg_t | 52 |
| DateTime | 54 |
| DS3231 | 55 |
| EventEmitter | |
| Provides a static method for emitting events | 65 |
| EventLog | |
| Represents a single event log entry | 67 |
| EventManager | |
| Manages the event logging system | 69 |
| EventManagerImpl | |
| Implementation of the <code>EventManager</code> class | 75 |
| FileHandle | 77 |
| Frame | 78 |
| HMC5883L | 80 |
| HMC5883LWrapper | 83 |
| INA3221 | 85 |
| ISensor | 103 |
| LoRaClass | 105 |
| INA3221::masken_reg_t | 133 |
| MPU6050Wrapper | 136 |
| NMEAData | 138 |
| PowerManager | 140 |
| Print | 146 |
| SensorWrapper | |
| Manages different sensor types and provides a unified interface for accessing sensor data | 166 |

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

| | | |
|--|---|-----|
| build_number.h | | 171 |
| includes.h | | 172 |
| main.cpp | | 346 |
| lib/pin_config.cpp | | 281 |
| lib/pin_config.h | | 283 |
| lib/utils.cpp | Implements utility functions for the Kabisat firmware | 340 |
| lib/utils.h | | 343 |
| lib/clock/DS3231.cpp | | 173 |
| lib/clock/DS3231.h | | 176 |
| lib/comms/communication.cpp | | 192 |
| lib/comms/communication.h | | 195 |
| lib/comms/frame.cpp | Implements functions for encoding, decoding, building, and processing Frames | 206 |
| lib/comms/protocol.h | | 241 |
| lib/comms/receive.cpp | Implements functions for receiving and processing data, including LoRa and UART input | 248 |
| lib/comms/send.cpp | Implements functions for sending data, including LoRa messages and Frames | 251 |
| lib/comms/utils_converters.cpp | Implements utility functions for converting between different data types | 254 |
| lib/comms/commands/clock_commands.cpp | | 177 |
| lib/comms/commands/commands.cpp | | 180 |
| lib/comms/commands/commands.h | | 181 |
| lib/comms/commands/diagnostic_commands.cpp | | 184 |
| lib/comms/commands/event_commands.cpp | | 185 |
| lib/comms/commands/gps_commands.cpp | | 187 |
| lib/comms/commands/power_commands.cpp | | 190 |
| lib/comms/LoRa/LoRa-RP2040.cpp | | 214 |
| lib/comms/LoRa/LoRa-RP2040.h | | 231 |
| lib/comms/LoRa/Print.cpp | | 234 |
| lib/comms/LoRa/Print.h | | 239 |
| lib/eventman/event_manager.cpp | Implements the event management system for the Kabisat firmware | 259 |
| lib/eventman/event_manager.h | | 265 |

| | |
|--|-----|
| lib/location/gps_collector.cpp | 272 |
| lib/location/gps_collector.h | 276 |
| lib/location/NMEA/NMEA_data.cpp | 278 |
| lib/location/NMEA/NMEA_data.h | 279 |
| lib/powerman/PowerManager.cpp | 301 |
| lib/powerman/PowerManager.h | 302 |
| lib/powerman/INA3221/INA3221.cpp | 290 |
| lib/powerman/INA3221/INA3221.h | 294 |
| lib/sensors/ISensor.cpp | |
| Implements the SensorWrapper class for managing different sensor types | 324 |
| lib/sensors/ISensor.h | 325 |
| lib/sensors/BH1750/BH1750.cpp | 304 |
| lib/sensors/BH1750/BH1750.h | 305 |
| lib/sensors/BH1750/BH1750_WRAPPER.cpp | 307 |
| lib/sensors/BH1750/BH1750_WRAPPER.h | 309 |
| lib/sensors/BME280/BME280.cpp | 310 |
| lib/sensors/BME280/BME280.h | 314 |
| lib/sensors/BME280/BME280_WRAPPER.cpp | 317 |
| lib/sensors/BME280/BME280_WRAPPER.h | 318 |
| lib/sensors/HMC5883L/HMC5883L.cpp | 319 |
| lib/sensors/HMC5883L/HMC5883L.h | 320 |
| lib/sensors/HMC5883L/HMC5883L_WRAPPER.cpp | 321 |
| lib/sensors/HMC5883L/HMC5883L_WRAPPER.h | 323 |
| lib/sensors/MPU6050/MPU6050.cpp | 328 |
| lib/sensors/MPU6050/MPU6050.h | 329 |
| lib/sensors/MPU6050/MPU6050_WRAPPER.cpp | 329 |
| lib/sensors/MPU6050/MPU6050_WRAPPER.h | 329 |
| lib/storage/storage.cpp | |
| Implements file system operations for the Kabisat firmware | 331 |
| lib/storage/storage.h | 336 |

Chapter 6

Topic Documentation

6.1 Clock Management Commands

Commands for managing system time and clock settings.

Functions

- `Frame handleTime (const std::string ¶m, OperationType operationType)`
Handler for getting and setting system time.
- `Frame handleTimezoneOffset (const std::string ¶m, OperationType operationType)`
Handler for getting and setting timezone offset.
- `Frame handleClockSyncInterval (const std::string ¶m, OperationType operationType)`
Handler for getting and setting clock synchronization interval.
- `Frame handleGetLastSyncTime (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 handleTime()

```
Frame handleTime (
    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

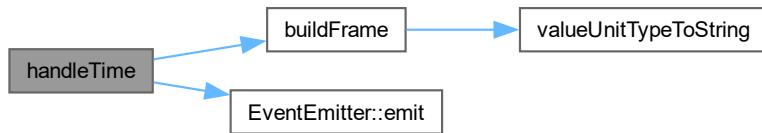
`Frame` containing success/error and current time or confirmation

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

Command Command ID: 3.0

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

Here is the call graph for this function:



6.1.2.2 handleTimezoneOffset()

```
Frame handleTimezoneOffset (
    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 88 of file [clock_commands.cpp](#).

Here is the call graph for this function:

**6.1.2.3 handleClockSyncInterval()**

```
Frame handleClockSyncInterval (
    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 130 of file [clock_commands.cpp](#).

Here is the call graph for this function:



6.1.2.4 handleGetLastSyncTime()

```
Frame handleGetLastSyncTime (
    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 167 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]
```

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

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 executeCommand (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 executeCommand()

```
Frame executeCommand (
    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

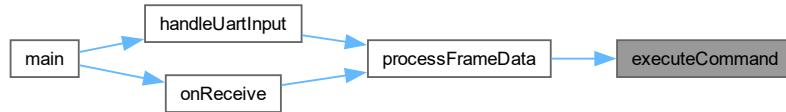
Looks up the command handler in commandHandlers map and executes it

Definition at line 59 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)), handleListCommands},
    {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(1)), handleGetBuildVersion},
    {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(9)), handleEnterBootloaderMode},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(0)), handleGetPowerManagerIDs},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(2)), handleGetVoltageBattery},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(3)), handleGetVoltage5V},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(4)), handleGetCurrentChargeUSB},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(5)), handleGetCurrentChargeSolar},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(6)), handleGetCurrentChargeTotal},
    {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(7)), handleGetCurrentDraw},
    {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(0)), handleTime},
    {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(1)), handleTimezoneOffset},
}
```

```

{{(static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(2)), handleClockSyncInterval},
{{(static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(3)), handleGetLastSyncTime},
{{(static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(1)), handleGetLastEvents},
{{(static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(2)), handleGetEventCount},
{{(static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(1)), handleGPSPowerStatus},
{{(static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(2)), handleEnableGPSTransparentMode},
{{(static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(3)), handleGetRMCData},
{{(static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(4)), handleGetGGAData},
}

```

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 handleListCommands** (const std::string ¶m, OperationType operationType)
Handler for listing all available commands on UART.
- **Frame handleGetBuildVersion** (const std::string ¶m, OperationType operationType)
Get firmware build version.
- **Frame handleEnterBootloaderMode** (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 handleListCommands()

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

Handler for listing all available commands on UART.

Parameters

| | |
|----------------------|-----------------------|
| <i>param</i> | Empty string expected |
| <i>operationType</i> | Must be GET |

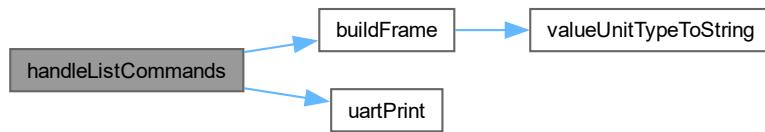
Returns

`Frame` containing success/error and command list

Command Command ID: 0

Definition at line 19 of file [diagnostic_commands.cpp](#).

Here is the call graph for this function:

**6.3.2.2 handleGetBuildVersion()**

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

Get firmware build version.

Parameters

| | |
|----------------------------|-----------------------|
| <code>param</code> | Empty string expected |
| <code>operationType</code> | Must be GET |

Returns

`Frame` containing build number

Command Command ID: 1

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

Here is the call graph for this function:

**6.3.2.3 handleEnterBootloaderMode()**

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

Reboot system to USB firmware loader.

Parameters

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

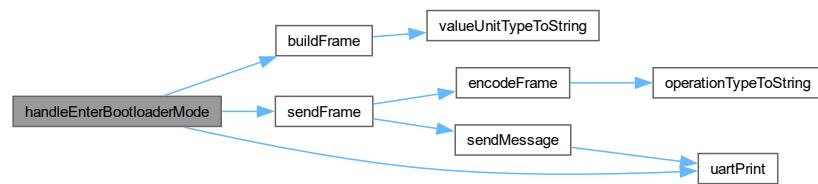
Returns

[Frame](#) with operation result

Command Command ID: 2

Definition at line 70 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 handleGetLastEvents (const std::string ¶m, OperationType operationType)`
Handler for retrieving last N events from the event log.
- `Frame handleGetEventCount (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 handleGetLastEvents()

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

Handler for retrieving last N events from the event log.

Parameters

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

Returns

[Frame](#) containing:

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

Note

KBST;0;GET;5;1;20;TSBK Returns up to 20 most recent events

[Command](#) Command ID: 5.1

Definition at line 29 of file [event_commands.cpp](#).

Here is the call graph for this function:

**6.4.2.2 handleGetEventCount()**

```
Frame handleGetEventCount (
    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

Command Command ID: 5.2

Definition at line 82 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 handleGPSPowerStatus** (const std::string ¶m, **OperationType** operationType)
Handler for controlling GPS module power state.
- **Frame handleEnableGPSTransparentMode** (const std::string ¶m, **OperationType** operationType)
Handler for enabling GPS transparent mode (UART pass-through)
- **Frame handleGetRMCDData** (const std::string ¶m, **OperationType** operationType)
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- **Frame handleGetGGAData** (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 handleGPSPowerStatus()

```

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

```

Handler for controlling GPS module power state.

Parameters

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

Returns

[Frame](#) containing:

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

Note

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

Return current GPS module power state: ON/OFF

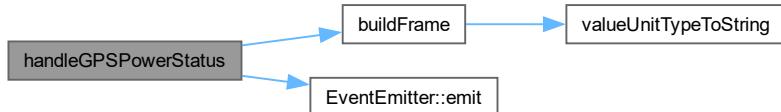
KBST;0;SET;7;1;POWER;TSBK

POWER - 0 - OFF, 1 - ON

Command Command ID: 7.1

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

Here is the call graph for this function:

**6.5.2.2 handleEnableGPSTransparentMode()**

```
Frame handleEnableGPSTransparentMode (
    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

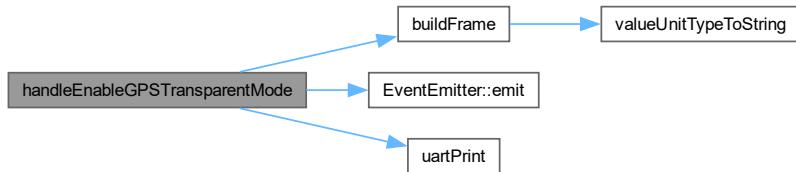
Enters a pass-through mode where UART communication is bridged directly to GPS

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

Command Command ID: 7.2

Definition at line [74](#) of file [gps_commands.cpp](#).

Here is the call graph for this function:

**6.5.2.3 handleGetRMCData()**

```
Frame handleGetRMCData (
    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 171 of file [gps_commands.cpp](#).

Here is the call graph for this function:



6.5.2.4 handleGetGGAData()

```
Frame handleGetGGAData (
    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 210 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 handleGetPowerManagerIDs** (const std::string ¶m, **OperationType** operationType)
Handler for retrieving Power Manager IDs.
- **Frame handleGetVoltageBattery** (const std::string ¶m, **OperationType** operationType)
Handler for getting battery voltage.
- **Frame handleGetVoltage5V** (const std::string ¶m, **OperationType** operationType)
Handler for getting 5V rail voltage.
- **Frame handleGetCurrentChargeUSB** (const std::string ¶m, **OperationType** operationType)
Handler for getting USB charge current.
- **Frame handleGetCurrentChargeSolar** (const std::string ¶m, **OperationType** operationType)
Handler for getting solar panel charge current.
- **Frame handleGetCurrentChargeTotal** (const std::string ¶m, **OperationType** operationType)
Handler for getting total charge current.
- **Frame handleGetCurrentDraw** (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 handleGetPowerManagerIDs()

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

Handler for retrieving Power Manager IDs.

Parameters

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

Returns

Frame containing:

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

Note

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

Command Command ID: 2.0

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

Here is the call graph for this function:

**6.6.2.2 handleGetVoltageBattery()**

```
Frame handleGetVoltageBattery (
    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

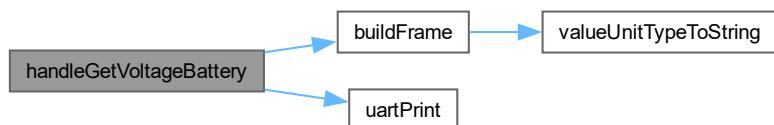
Note

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

Command Command ID: 2.2

Definition at line 47 of file [power_commands.cpp](#).

Here is the call graph for this function:



6.6.2.3 handleGetVoltage5V()

```
Frame handleGetVoltage5V (
    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

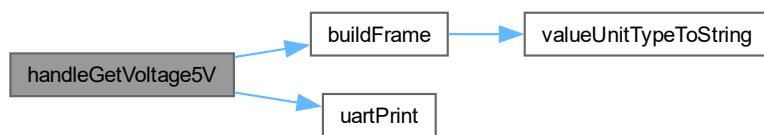
Note

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

Command Command ID: 2.3

Definition at line [75](#) of file [power_commands.cpp](#).

Here is the call graph for this function:



6.6.2.4 handleGetCurrentChargeUSB()

```
Frame handleGetCurrentChargeUSB (
    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

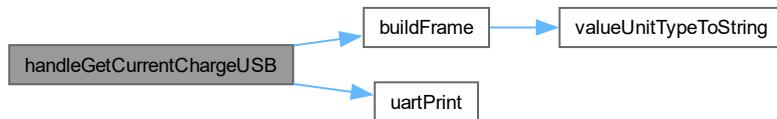
Note

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

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 handleGetCurrentChargeSolar()**

```
Frame handleGetCurrentChargeSolar (
    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

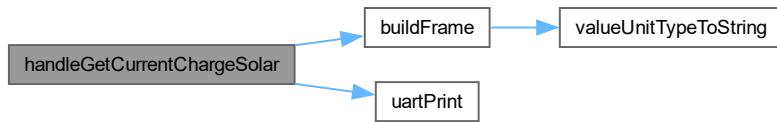
Note

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

Command Command ID: 2.5

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

Here is the call graph for this function:

**6.6.2.6 handleGetCurrentChargeTotal()**

```
Frame handleGetCurrentChargeTotal (
    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

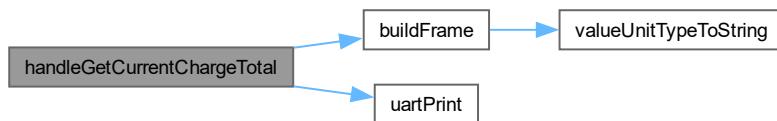
Note

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

Command Command ID: 2.6

Definition at line 159 of file [power_commands.cpp](#).

Here is the call graph for this function:



6.6.2.7 handleGetCurrentDraw()

```
Frame handleGetCurrentDraw (
    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

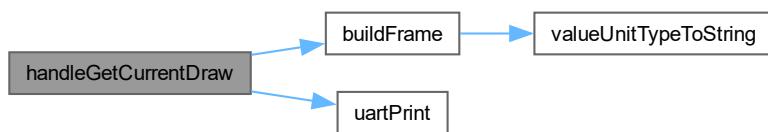
Note

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

Command Command ID: 2.7

Definition at line 187 of file [power_commands.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 `readLightLevel` ()

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 readLightLevel()

```
float BH1750::readLightLevel ()
```

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](#).

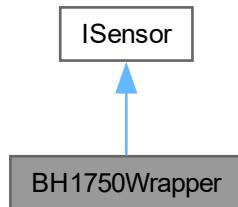
The documentation for this class was generated from the following files:

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

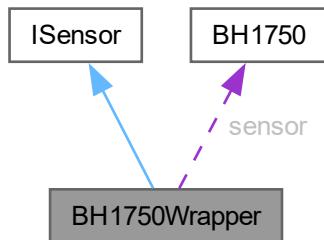
7.2 BH1750Wrapper Class Reference

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

Inheritance diagram for BH1750Wrapper:



Collaboration diagram for BH1750Wrapper:



Public Member Functions

- `BH1750Wrapper ()`
- `int get_i2c_addr ()`
- `bool init () override`
- `float readData (SensorDataTypelIdentifier type) override`
- `bool isInitialized () const override`
- `SensorType getType () 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 `readData()`

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

Implements [ISensor](#).

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

7.2.3.4 `isInitialized()`

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

Implements [ISensor](#).

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

7.2.3.5 `getType()`

```
SensorType BH1750Wrapper::getType () 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](#).

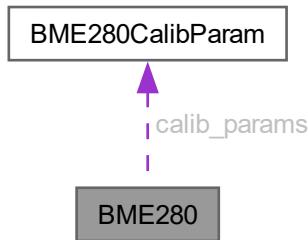
The documentation for this class was generated from the following files:

- 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](#).

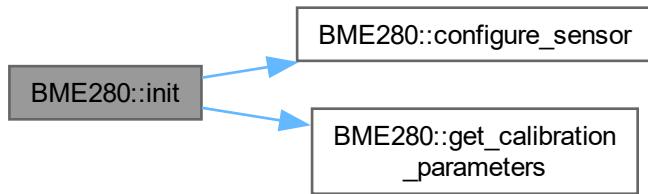
7.3.3 Member Function Documentation

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](#).

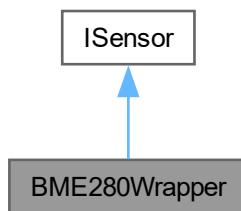
The documentation for this struct was generated from the following file:

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

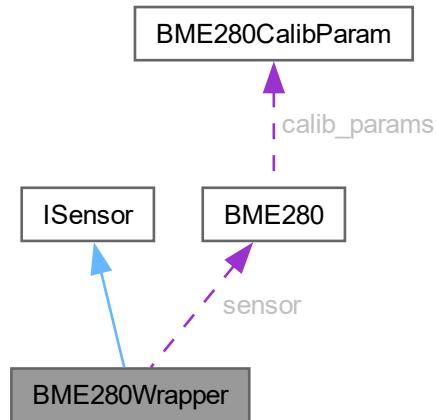
7.5 BME280Wrapper Class Reference

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

Inheritance diagram for BME280Wrapper:



Collaboration diagram for BME280Wrapper:



Public Member Functions

- [BME280Wrapper \(i2c_inst_t *i2c\)](#)
- bool [init \(\)](#) override
- float [readData \(SensorDataTypelIdentifier type\)](#) override
- bool [isInitialized \(\)](#) const override
- [SensorType getType \(\)](#) 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 readData()

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

Implements [ISensor](#).

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

7.5.3.3 isInitialized()

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

Implements [ISensor](#).

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

7.5.3.4 getType()

```
SensorType BME280Wrapper::getType () 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

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

Definition at line 76 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 77 of file [INA3221.h](#).

7.6.2.2 mode_bus_en

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

Definition at line 78 of file [INA3221.h](#).

7.6.2.3 mode_continious_en

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

Definition at line 79 of file [INA3221.h](#).

7.6.2.4 shunt_conv_time

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

Definition at line 80 of file [INA3221.h](#).

7.6.2.5 bus_conv_time

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

Definition at line 81 of file [INA3221.h](#).

7.6.2.6 avg_mode

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

Definition at line 82 of file [INA3221.h](#).

7.6.2.7 ch3_en

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

Definition at line 83 of file [INA3221.h](#).

7.6.2.8 ch2_en

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

Definition at line 84 of file [INA3221.h](#).

7.6.2.9 ch1_en

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

Definition at line 85 of file [INA3221.h](#).

7.6.2.10 reset

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

Definition at line 86 of file [INA3221.h](#).

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

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

7.7 DateTime Struct Reference

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

Public Attributes

- uint16_t [year](#)
- uint8_t [month](#)
- uint8_t [day](#)
- uint8_t [hour](#)
- uint8_t [minute](#)
- uint8_t [second](#)
- std::string [weekday](#)

7.7.1 Detailed Description

Definition at line 9 of file [DS3231.h](#).

7.7.2 Member Data Documentation

7.7.2.1 year

```
uint16_t DateTime::year
```

Definition at line 10 of file [DS3231.h](#).

7.7.2.2 month

```
uint8_t DateTime::month
```

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

7.7.2.3 day

```
uint8_t DateTime::day
```

Definition at line 12 of file [DS3231.h](#).

7.7.2.4 hour

```
uint8_t DateTime::hour
```

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

7.7.2.5 minute

```
uint8_t DateTime::minute
```

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

7.7.2.6 second

```
uint8_t DateTime::second
```

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

7.7.2.7 weekday

```
std::string DateTime::weekday
```

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

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

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

7.8 DS3231 Class Reference

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

Public Member Functions

- [DS3231](#) (*i2c_inst_t *i2c*, *uint8_t address=0x68*)
- bool [setTime](#) (*uint8_t sec*, *uint8_t min*, *uint8_t hour*, *uint8_t weekday*, *uint8_t day*, *uint8_t month*, *uint16_t year*)
- bool [getTime](#) (*uint8_t &sec*, *uint8_t &min*, *uint8_t &hour*, *std::string &weekday*, *uint8_t &day*, *uint8_t &month*, *uint16_t &year*)
- std::string [getTimeString](#) ()
- bool [setTimeUnix](#) (*uint32_t unixTime*)
- *uint32_t getTimeUnix* ()
- [DateTime getDate](#) ()
- *uint64_t getTimel* ()
- void [setTimezoneOffset](#) (*int16_t offsetMinutes*)
- *int16_t getTimezoneOffset* () const
- *uint32_t getTimeUnixLocal* ()
- [DateTime getDateLocal](#) ()
- void [setClockSyncInterval](#) (*uint32_t intervalSeconds*)
- *uint32_t getClockSyncInterval* () const
- void [setLastSyncTime](#) (*uint32_t unixTime*)
- *uint32_t getLastSyncTime* () const

Static Public Attributes

- static const std::array< std::string, 7 > WEEKDAYS

Private Member Functions

- uint8_t `bcd2bin` (uint8_t val)
- uint8_t `bin2bcd` (uint8_t val)
- std::string `preZero` (uint8_t val)
- `DateTime applyTimezone` (const `DateTime` &utc, int16_t offsetMinutes)

Static Private Member Functions

- static uint32_t `dateTimeToUnix` (const `DateTime` &dt)
- static `DateTime unixToDate` (uint32_t unixTime)

Private Attributes

- i2c_inst_t * `i2c`
- uint8_t `address`
- int16_t `timezoneOffset` = 0
- uint32_t `syncInterval` = 86400
- uint32_t `lastSyncTime` = 0

Static Private Attributes

- static constexpr uint8_t `RTC_REGISTER` = 0x00

7.8.1 Detailed Description

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

7.8.2 Constructor & Destructor Documentation

7.8.2.1 DS3231()

```
DS3231::DS3231 (
    i2c_inst_t * i2c,
    uint8_t address = 0x68)
```

Definition at line 9 of file [DS3231.cpp](#).

7.8.3 Member Function Documentation

7.8.3.1 setTime()

```
bool DS3231::setTime (
    uint8_t sec,
    uint8_t min,
    uint8_t hour,
    uint8_t weekday,
    uint8_t day,
    uint8_t month,
    uint16_t year)
```

Definition at line 12 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3.2 getTime()

```
bool DS3231::getTime (
    uint8_t & sec,
    uint8_t & min,
    uint8_t & hour,
    std::string & weekday,
    uint8_t & day,
    uint8_t & month,
    uint16_t & year)
```

Definition at line 29 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3.3 `getTimeString()`

```
std::string DS3231::getTimeString ()
```

Definition at line 53 of file [DS3231.cpp](#).

Here is the call graph for this function:



7.8.3.4 `setTimeUnix()`

```
bool DS3231::setTimeUnix (
    uint32_t unixTime)
```

Definition at line 87 of file [DS3231.cpp](#).

Here is the call graph for this function:



7.8.3.5 `getTimeUnix()`

```
uint32_t DS3231::getTimeUnix ()
```

Definition at line 99 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.3.6 `getDateTime()`

```
DateTime DS3231::getDateTime ()
```

Definition at line 104 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:

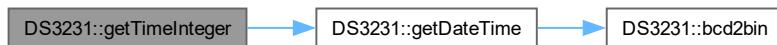


7.8.3.7 getTimeInteger()

```
uint64_t DS3231::getTimeInteger ()
```

Definition at line 128 of file [DS3231.cpp](#).

Here is the call graph for this function:



7.8.3.8 setTimezoneOffset()

```
void DS3231::setTimezoneOffset (
    int16_t offsetMinutes) [inline]
```

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

7.8.3.9 `getTimezoneOffset()`

```
int16_t DS3231::getTimezoneOffset () const [inline]
```

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

7.8.3.10 `getTimeUnixLocal()`

```
uint32_t DS3231::getTimeUnixLocal ()
```

Definition at line 165 of file [DS3231.cpp](#).

Here is the call graph for this function:

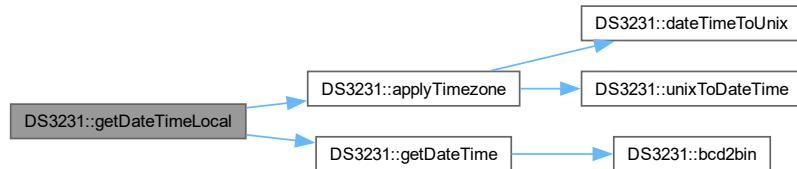


7.8.3.11 `getDateTimeLocal()`

```
DateTime DS3231::getDateTimeLocal ()
```

Definition at line 170 of file [DS3231.cpp](#).

Here is the call graph for this function:



7.8.3.12 `setClockSyncInterval()`

```
void DS3231::setClockSyncInterval (
    uint32_t intervalSeconds) [inline]
```

Definition at line 46 of file [DS3231.h](#).

7.8.3.13 `getClockSyncInterval()`

```
uint32_t DS3231::getClockSyncInterval () const [inline]
```

Definition at line 47 of file [DS3231.h](#).

7.8.3.14 `setLastSyncTime()`

```
void DS3231::setLastSyncTime (
    uint32_t unixTime) [inline]
```

Definition at line 49 of file [DS3231.h](#).

7.8.3.15 `getLastSyncTime()`

```
uint32_t DS3231::getLastSyncTime () const [inline]
```

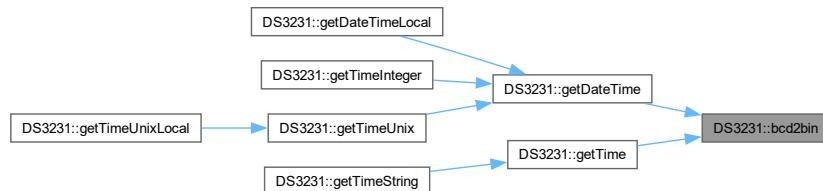
Definition at line 50 of file [DS3231.h](#).

7.8.3.16 `bcd2bin()`

```
uint8_t DS3231::bcd2bin (
    uint8_t val) [private]
```

Definition at line 72 of file [DS3231.cpp](#).

Here is the caller graph for this function:

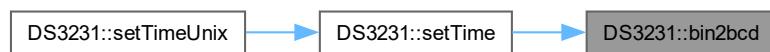


7.8.3.17 `bin2bcd()`

```
uint8_t DS3231::bin2bcd (
    uint8_t val) [private]
```

Definition at line 76 of file [DS3231.cpp](#).

Here is the caller graph for this function:



7.8.3.18 preZero()

```
std::string DS3231::preZero (
    uint8_t val) [private]
```

Definition at line 80 of file [DS3231.cpp](#).

Here is the caller graph for this function:

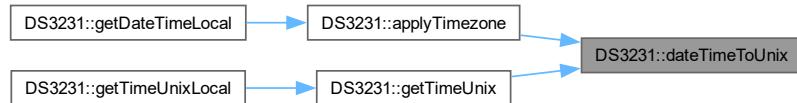


7.8.3.19 dateTimeToUnix()

```
uint32_t DS3231::dateTimeToUnix (
    const DateTime & dt) [static], [private]
```

Definition at line 138 of file [DS3231.cpp](#).

Here is the caller graph for this function:

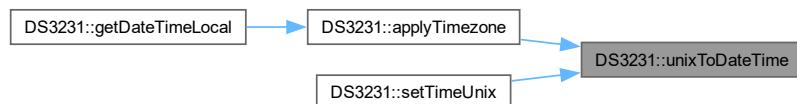


7.8.3.20 unixToDateTIme()

```
DateTime DS3231::unixToDateTIme (
    uint32_t unixTime) [static], [private]
```

Definition at line 149 of file [DS3231.cpp](#).

Here is the caller graph for this function:

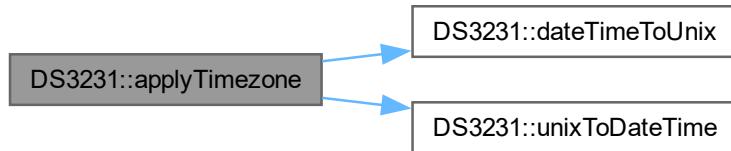


7.8.3.21 applyTimezone()

```
DateTime DS3231::applyTimezone (
    const DateTime & utc,
    int16_t offsetMinutes) [private]
```

Definition at line 175 of file [DS3231.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.4 Member Data Documentation

7.8.4.1 WEEKDAYS

```
const std::array< std::string, 7 > DS3231::WEEKDAYS [static]
```

Initial value:

```
= {
    "SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"
}
```

Definition at line 5 of file [DS3231.h](#).

7.8.4.2 i2c

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

Definition at line 54 of file [DS3231.h](#).

7.8.4.3 address

```
uint8_t DS3231::address [private]
```

Definition at line 55 of file [DS3231.h](#).

7.8.4.4 RTC_REGISTER

```
uint8_t DS3231::RTC_REGISTER = 0x00 [static], [constexpr], [private]
```

Definition at line 56 of file [DS3231.h](#).

7.8.4.5 timezoneOffset

```
int16_t DS3231::timezoneOffset = 0 [private]
```

Definition at line 58 of file [DS3231.h](#).

7.8.4.6 syncInterval

```
uint32_t DS3231::syncInterval = 86400 [private]
```

Definition at line 59 of file [DS3231.h](#).

7.8.4.7 lastSyncTime

```
uint32_t DS3231::lastSyncTime = 0 [private]
```

Definition at line 60 of file [DS3231.h](#).

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

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

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 214 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

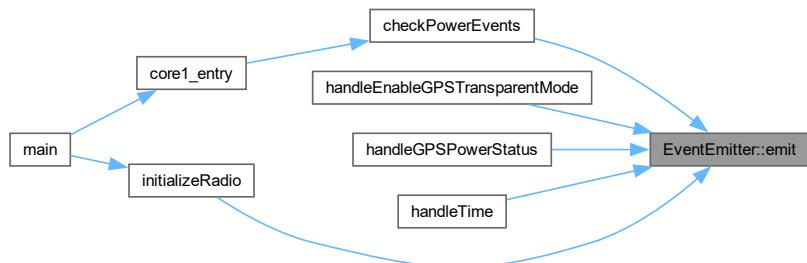
| | |
|--------------|-----------------------|
| <i>group</i> | The event group. |
| <i>event</i> | The event identifier. |

Template Parameters

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

Definition at line 223 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 [toString \(\) 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 [73](#) of file [event_manager.h](#).

7.10.2 Member Function Documentation

7.10.2.1 [toString\(\)](#)

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

Converts the [EventLog](#) to a string representation.

Returns

A string representation of the [EventLog](#).

Definition at line [84](#) 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 [75](#) of file [event_manager.h](#).

7.10.3.2 timestamp

```
uint32_t EventLog::timestamp
```

Unix timestamp or system time.

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

7.10.3.3 group

```
uint8_t EventLog::group
```

Event group identifier.

Definition at line [77](#) of file [event_manager.h](#).

7.10.3.4 event

```
uint8_t EventLog::event
```

Specific event identifier.

Definition at line [78](#) of file [event_manager.h](#).

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

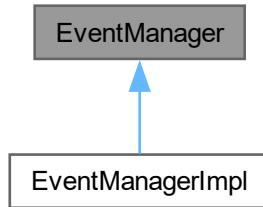
- lib/eventman/[event_manager.h](#)

7.11 EventManager Class Reference

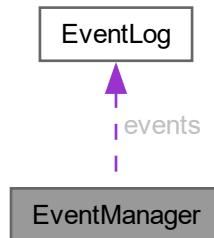
Manages the event logging system.

```
#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 logEvent (uint8_t group, uint8_t event)`
Logs an event.
- `const EventLog & getEvent (size_t index) const`
Retrieves an event from the event buffer.
- `size_t getEventCount () const`

- Gets the number of events in the buffer.
- virtual bool [saveToStorage \(\)=0](#)
Saves the events to storage.
- virtual bool [loadFromStorage \(\)=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 98 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 104 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 122 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 logEvent()

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

Logs an event.

Logs an event to the event buffer.

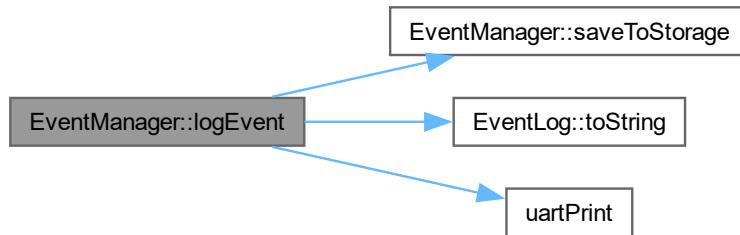
Parameters

| | |
|--------------|-----------------------|
| <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 `getEvent()`

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

Retrieves an event from the event buffer.

Parameters

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

Returns

A const reference to the `EventLog` at the specified index.

Parameters

| | |
|--------------------|-------------------------------------|
| <code>index</code> | 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 `getEventCount()`

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

Gets the number of events in the buffer.

Returns

The number of events in the buffer.

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

7.11.3.5 `saveToStorage()`

```
virtual bool EventManager::saveToStorage () [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 `loadFromStorage()`

```
virtual bool EventManager::loadFromStorage () [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 159 of file [event_manager.h](#).

7.11.4.2 eventCount

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

Number of events in the buffer.

Definition at line 160 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 161 of file [event_manager.h](#).

7.11.4.4 eventMutex

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

Mutex for protecting the event buffer.

Definition at line 162 of file [event_manager.h](#).

7.11.4.5 nextEventId

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

Next event ID.

Definition at line 163 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 164 of file [event_manager.h](#).

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

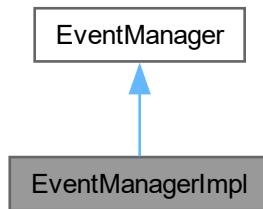
- lib/eventman/[event_manager.h](#)
- lib/eventman/[event_manager.cpp](#)

7.12 EventManagerImpl Class Reference

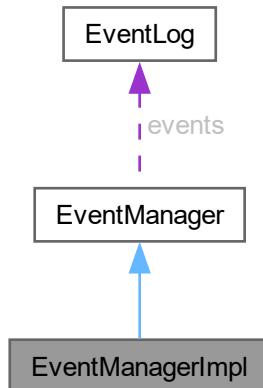
Implementation of the [EventManager](#) class.

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

Inheritance diagram for EventManagerImpl:



Collaboration diagram for EventManagerImpl:



Public Member Functions

- [EventManagerImpl \(\)](#)
Constructor for the EventManagerImpl.
- [bool saveToStorage \(\) override](#)
Saves the events to storage.
- [bool loadFromStorage \(\) 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 [logEvent \(uint8_t group, uint8_t event\)](#)
Logs an event.
- const [EventLog & getEvent \(size_t index\) const](#)
Retrieves an event from the event buffer.
- size_t [getEventCount \(\) 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 172 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 178 of file [event_manager.h](#).

Here is the call graph for this function:



7.12.3 Member Function Documentation

7.12.3.1 saveToStorage()

```
bool EventManagerImpl::saveToStorage () [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 187 of file [event_manager.h](#).

7.12.3.2 loadFromStorage()

```
bool EventManagerImpl::loadFromStorage () [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 198 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 15 of file [storage.h](#).

7.13.2 Member Data Documentation

7.13.2.1 fd

```
int FileHandle::fd
```

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

7.13.2.2 is_open

```
bool FileHandle::is_open
```

Definition at line 17 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 65 of file [protocol.h](#).

7.14.2 Member Data Documentation

7.14.2.1 header

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

Definition at line [66](#) of file [protocol.h](#).

7.14.2.2 direction

```
uint8_t Frame::direction
```

Definition at line [67](#) of file [protocol.h](#).

7.14.2.3 operationType

```
OperationType Frame::operationType
```

Definition at line [68](#) of file [protocol.h](#).

7.14.2.4 group

```
uint8_t Frame::group
```

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

7.14.2.5 command

```
uint8_t Frame::command
```

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

7.14.2.6 value

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

Definition at line [71](#) of file [protocol.h](#).

7.14.2.7 unit

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

Definition at line [72](#) of file [protocol.h](#).

7.14.2.8 footer

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

Definition at line 73 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 [writeRegister](#) (*uint8_t* *reg*, *uint8_t* *value*)
- bool [readRegisters](#) (*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 writeRegister()

```
bool HMC5883L::writeRegister (
    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 `readRegisters()`

```
bool HMC5883L::readRegisters (
    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](#).

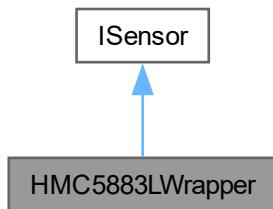
The documentation for this class was generated from the following files:

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

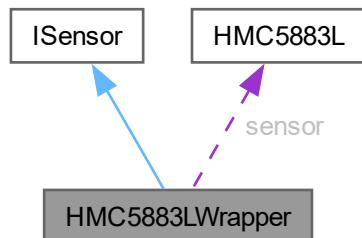
7.16 HMC5883LWrapper Class Reference

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

Inheritance diagram for HMC5883LWrapper:



Collaboration diagram for HMC5883LWrapper:



Public Member Functions

- [HMC5883LWrapper \(i2c_inst_t *i2c\)](#)
- bool [init \(\)](#) override
- float [readData \(SensorDataTypelIdentifier type\)](#) override
- bool [isInitialized \(\)](#) const override
- [SensorType getType \(\)](#) 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 `readData()`

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

Implements [ISensor](#).

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

7.16.3.3 `isInitialized()`

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

Implements [ISensor](#).

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

7.16.3.4 getType()

```
SensorType HMC5883LWrapper::getType () 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](#).

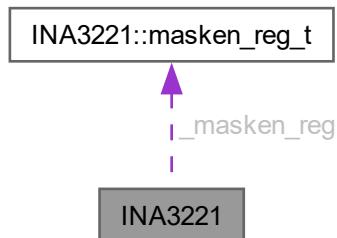
The documentation for this class was generated from the following files:

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

7.17 INA3221 Class Reference

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

Collaboration diagram for INA3221:



Classes

- struct `conf_reg_t`
- struct `masken_reg_t`

Public Member Functions

- `INA3221 (ina3221_addr_t addr, i2c_inst_t *i2c)`
- `bool begin ()`
- `void setShuntRes (uint32_t res_ch1, uint32_t res_ch2, uint32_t res_ch3)`
- `void setFilterRes (uint32_t res_ch1, uint32_t res_ch2, uint32_t res_ch3)`
- `void setAddr (ina3221_addr_t addr)`
- `uint16_t getReg (ina3221_reg_t reg)`
- `void reset ()`
- `void setModePowerDown ()`
- `void setModeContinious ()`
- `void setModeTriggered ()`
- `void setShuntMeasEnable ()`
- `void setShuntMeasDisable ()`
- `void setBusMeasEnable ()`
- `void setBusMeasDisable ()`
- `void setAveragingMode (ina3221_avg_mode_t mode)`
- `void setBusConversionTime (ina3221_conv_time_t convTime)`
- `void setShuntConversionTime (ina3221_conv_time_t convTime)`
- `void setCritAlertLimit (ina3221_ch_t channel, int32_t voltageuV)`
- `void setWarnAlertLimit (ina3221_ch_t channel, int32_t voltageuV)`
- `void setPwrValidUpLimit (int16_t voltagemV)`
- `void setPwrValidLowLimit (int16_t voltagemV)`
- `void setShuntSumAlertLimit (int32_t voltagemV)`
- `void setCurrentSumAlertLimit (int32_t currentmA)`
- `void setWarnAlertLatchEnable ()`
- `void setWarnAlertLatchDisable ()`
- `void setCritAlertLatchEnable ()`
- `void setCritAlertLatchDisable ()`
- `void readFlags ()`
- `bool getTimingCtrlAlertFlag ()`
- `bool getPwrValidAlertFlag ()`
- `bool getCurrentSumAlertFlag ()`
- `bool getConversionReadyFlag ()`
- `uint16_t getManufID ()`
- `uint16_t getDieID ()`
- `void setChannelEnable (ina3221_ch_t channel)`
- `void setChannelDisable (ina3221_ch_t channel)`
- `void setWarnAlertShuntLimit (ina3221_ch_t channel, int32_t voltageuV)`
- `void setCritAlertShuntLimit (ina3221_ch_t channel, int32_t voltageuV)`
- `void setWarnAlertCurrentLimit (ina3221_ch_t channel, int32_t currentmA)`
- `void setCritAlertCurrentLimit (ina3221_ch_t channel, int32_t currentmA)`
- `void setCurrentSumEnable (ina3221_ch_t channel)`
- `void setCurrentSumDisable (ina3221_ch_t channel)`
- `int32_t getShuntVoltage (ina3221_ch_t channel)`
- `bool getWarnAlertFlag (ina3221_ch_t channel)`
- `bool getCritAlertFlag (ina3221_ch_t channel)`
- `int32_t estimateOffsetVoltage (ina3221_ch_t channel, uint32_t busVoltage)`
- `float getCurrent (ina3221_ch_t channel)`
- `float getCurrent_mA (ina3221_ch_t channel)`
- `float getCurrentCompensated (ina3221_ch_t channel)`
- `float getCurrentCompensated_mA (ina3221_ch_t channel)`
- `float getVoltage (ina3221_ch_t channel)`

Private Member Functions

- void [_read](#) (ina3221_reg_t reg, uint16_t *val)
- void [_write](#) (ina3221_reg_t reg, uint16_t *val)

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

Definition at line [73](#) of file [INA3221.h](#).

7.17.2 Constructor & Destructor Documentation

7.17.2.1 INA3221()

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

Definition at line [10](#) of file [INA3221.cpp](#).

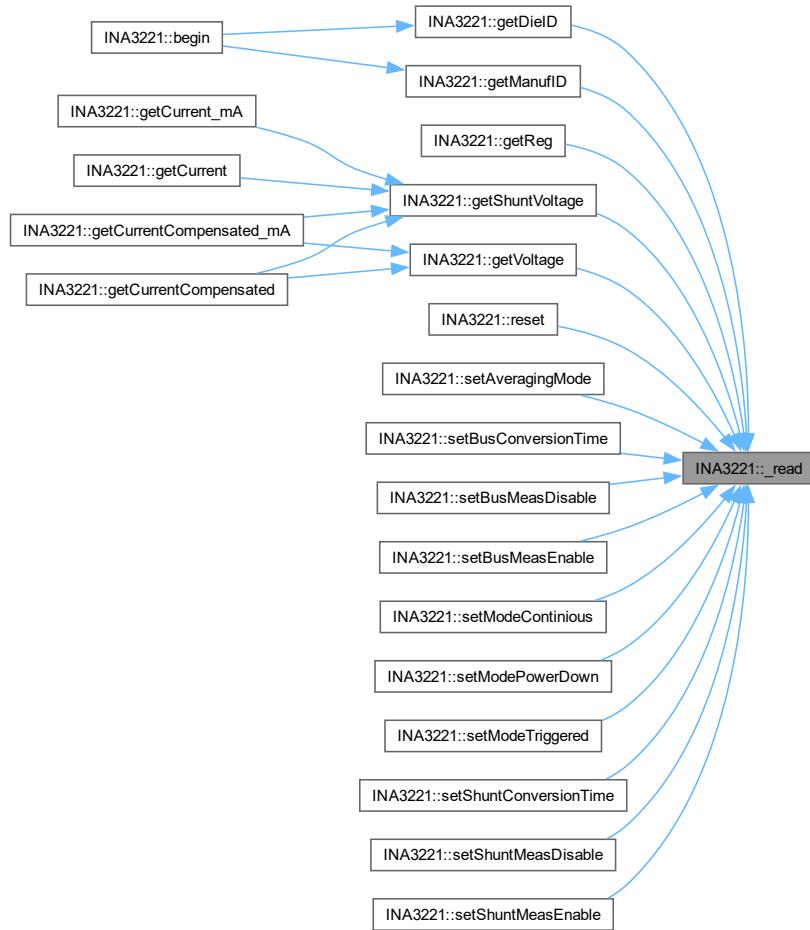
7.17.3 Member Function Documentation

7.17.3.1 _read()

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

Definition at line [42](#) of file [INA3221.cpp](#).

Here is the caller graph for this function:



7.17.3.2 `_write()`

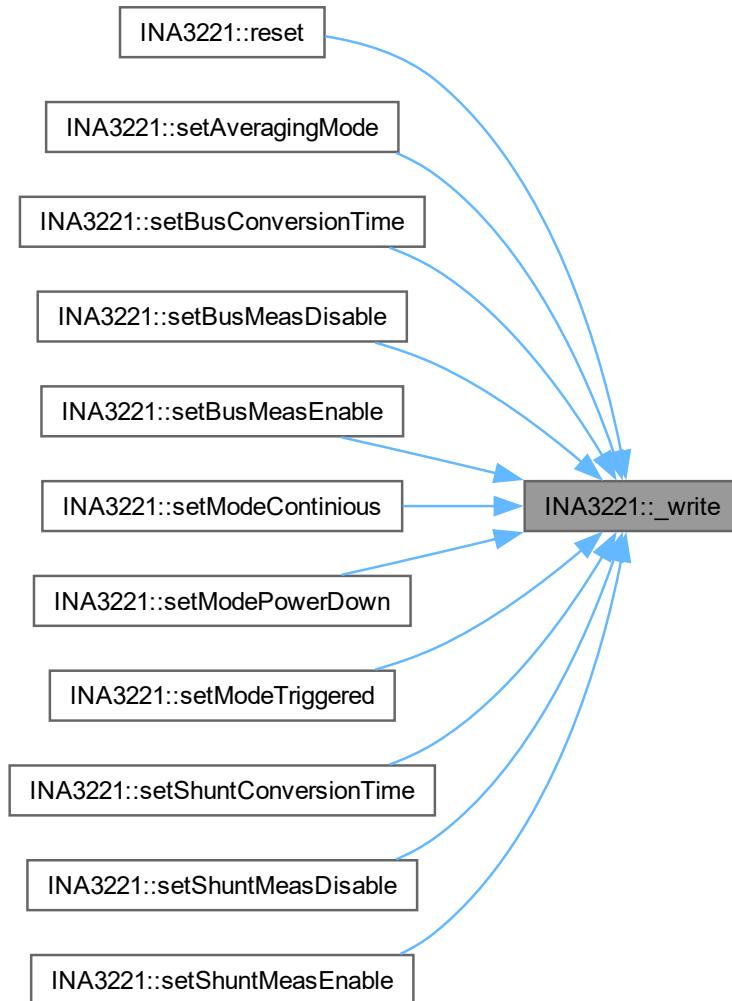
```

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

```

Definition at line 61 of file [INA3221.cpp](#).

Here is the caller graph for this function:

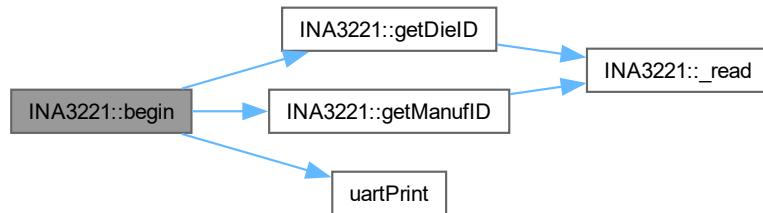


7.17.3.3 begin()

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

Definition at line 14 of file [INA3221.cpp](#).

Here is the call graph for this function:



7.17.3.4 `setShuntRes()`

```
void INA3221::setShuntRes (
    uint32_t res_ch1,
    uint32_t res_ch2,
    uint32_t res_ch3)
```

7.17.3.5 `setFilterRes()`

```
void INA3221::setFilterRes (
    uint32_t res_ch1,
    uint32_t res_ch2,
    uint32_t res_ch3)
```

7.17.3.6 `setAddr()`

```
void INA3221::setAddr (
    ina3221_addr_t addr) [inline]
```

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

7.17.3.7 `getReg()`

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

Definition at line 73 of file [INA3221.cpp](#).

Here is the call graph for this function:

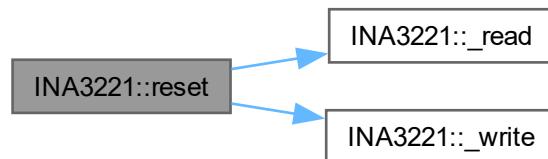


7.17.3.8 reset()

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

Definition at line 79 of file [INA3221.cpp](#).

Here is the call graph for this function:

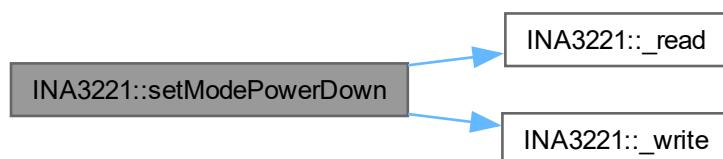


7.17.3.9 setModePowerDown()

```
void INA3221::setModePowerDown ()
```

Definition at line 87 of file [INA3221.cpp](#).

Here is the call graph for this function:

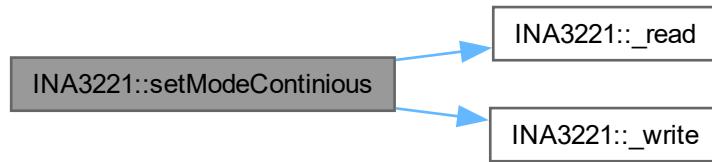


7.17.3.10 setModeContinious()

```
void INA3221::setModeContinious ()
```

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

Here is the call graph for this function:

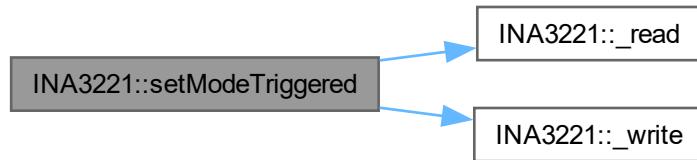


7.17.3.11 setModeTriggered()

```
void INA3221::setModeTriggered ()
```

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

Here is the call graph for this function:

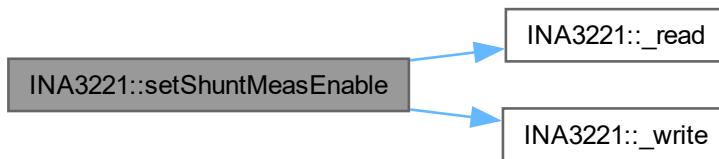


7.17.3.12 setShuntMeasEnable()

```
void INA3221::setShuntMeasEnable ()
```

Definition at line 112 of file [INA3221.cpp](#).

Here is the call graph for this function:

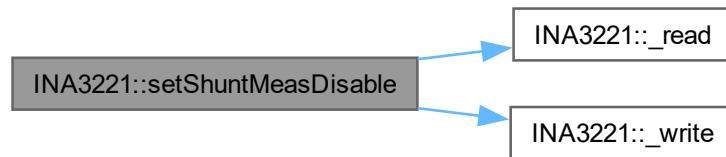


7.17.3.13 setShuntMeasDisable()

```
void INA3221::setShuntMeasDisable ()
```

Definition at line 120 of file [INA3221.cpp](#).

Here is the call graph for this function:

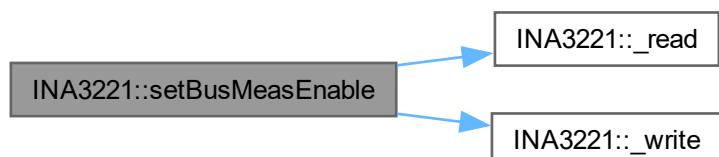


7.17.3.14 setBusMeasEnable()

```
void INA3221::setBusMeasEnable ()
```

Definition at line 128 of file [INA3221.cpp](#).

Here is the call graph for this function:

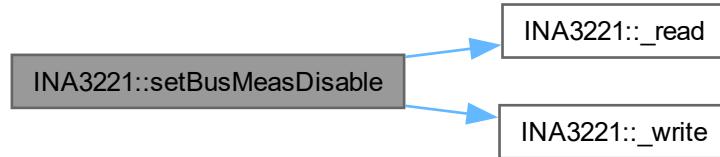


7.17.3.15 setBusMeasDisable()

```
void INA3221::setBusMeasDisable ()
```

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

Here is the call graph for this function:

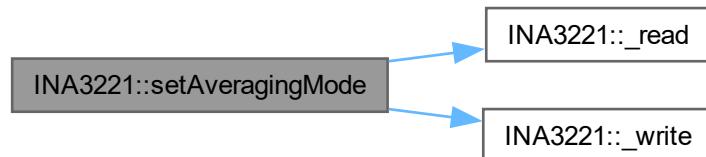


7.17.3.16 setAveragingMode()

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

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

Here is the call graph for this function:

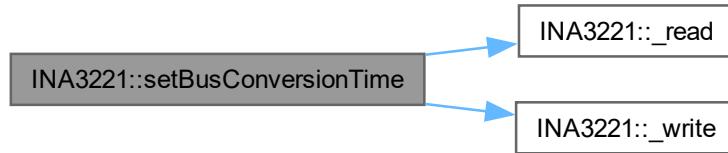


7.17.3.17 setBusConversionTime()

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

Definition at line 152 of file [INA3221.cpp](#).

Here is the call graph for this function:



7.17.3.18 setShuntConversionTime()

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

Definition at line 160 of file [INA3221.cpp](#).

Here is the call graph for this function:



7.17.3.19 setCritAlertLimit()

```
void INA3221::setCritAlertLimit (
    ina3221_ch_t channel,
    int32_t voltageuV)
```

7.17.3.20 setWarnAlertLimit()

```
void INA3221::setWarnAlertLimit (
    ina3221_ch_t channel,
    int32_t voltageuV)
```

7.17.3.21 setPwrValidUpLimit()

```
void INA3221::setPwrValidUpLimit (
    int16_t voltagemV)
```

7.17.3.22 setPwrValidLowLimit()

```
void INA3221::setPwrValidLowLimit (
    int16_t voltagemV)
```

7.17.3.23 setShuntSumAlertLimit()

```
void INA3221::setShuntSumAlertLimit (
    int32_t voltagemV)
```

7.17.3.24 setCurrentSumAlertLimit()

```
void INA3221::setCurrentSumAlertLimit (
    int32_t currentmA)
```

7.17.3.25 setWarnAlertLatchEnable()

```
void INA3221::setWarnAlertLatchEnable ()
```

7.17.3.26 setWarnAlertLatchDisable()

```
void INA3221::setWarnAlertLatchDisable ()
```

7.17.3.27 setCritAlertLatchEnable()

```
void INA3221::setCritAlertLatchEnable ()
```

7.17.3.28 setCritAlertLatchDisable()

```
void INA3221::setCritAlertLatchDisable ()
```

7.17.3.29 readFlags()

```
void INA3221::readFlags ()
```

7.17.3.30 getTimingCtrlAlertFlag()

```
bool INA3221::getTimingCtrlAlertFlag ()
```

7.17.3.31 `getPwrValidAlertFlag()`

```
bool INA3221::getPwrValidAlertFlag ()
```

7.17.3.32 `getCurrentSumAlertFlag()`

```
bool INA3221::getCurrentSumAlertFlag ()
```

7.17.3.33 `getConversionReadyFlag()`

```
bool INA3221::getConversionReadyFlag ()
```

7.17.3.34 `getManufID()`

```
uint16_t INA3221::getManufID ()
```

Definition at line 168 of file [INA3221.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.35 `getDieID()`

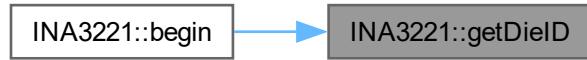
```
uint16_t INA3221::getDieID ()
```

Definition at line 174 of file [INA3221.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.36 `setChannelEnable()`

```
void INA3221::setChannelEnable (
    ina3221_ch_t channel)
```

7.17.3.37 `setChannelDisable()`

```
void INA3221::setChannelDisable (
    ina3221_ch_t channel)
```

7.17.3.38 `setWarnAlertShuntLimit()`

```
void INA3221::setWarnAlertShuntLimit (
    ina3221_ch_t channel,
    int32_t voltageuV)
```

7.17.3.39 `setCritAlertShuntLimit()`

```
void INA3221::setCritAlertShuntLimit (
    ina3221_ch_t channel,
    int32_t voltageuV)
```

7.17.3.40 setWarnAlertCurrentLimit()

```
void INA3221::setWarnAlertCurrentLimit (
    ina3221_ch_t channel,
    int32_t currentmA)
```

7.17.3.41 setCritAlertCurrentLimit()

```
void INA3221::setCritAlertCurrentLimit (
    ina3221_ch_t channel,
    int32_t currentmA)
```

7.17.3.42 setCurrentSumEnable()

```
void INA3221::setCurrentSumEnable (
    ina3221_ch_t channel)
```

7.17.3.43 setCurrentSumDisable()

```
void INA3221::setCurrentSumDisable (
    ina3221_ch_t channel)
```

7.17.3.44 getShuntVoltage()

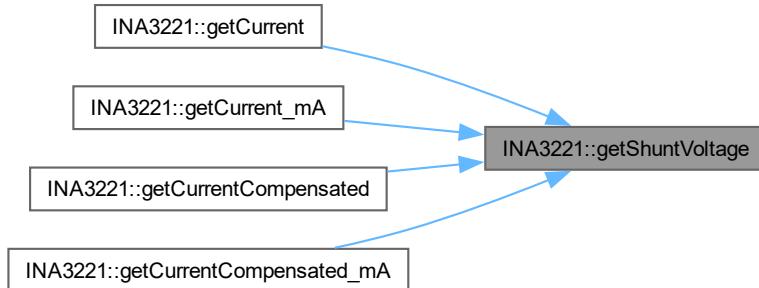
```
int32_t INA3221::getShuntVoltage (
    ina3221_ch_t channel)
```

Definition at line 180 of file [INA3221.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.3.45 `getWarnAlertFlag()`

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

7.17.3.46 `getCritAlertFlag()`

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

7.17.3.47 `estimateOffsetVoltage()`

```
int32_t INA3221::estimateOffsetVoltage (
    ina3221_ch_t channel,
    uint32_t busVoltage)
```

Definition at line 205 of file [INA3221.cpp](#).

Here is the caller graph for this function:

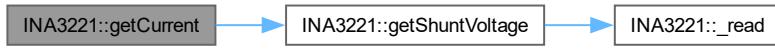


7.17.3.48 getCurrent()

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

Definition at line 228 of file [INA3221.cpp](#).

Here is the call graph for this function:

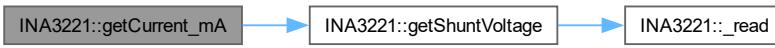


7.17.3.49 getCurrent_mA()

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

Definition at line 237 of file [INA3221.cpp](#).

Here is the call graph for this function:

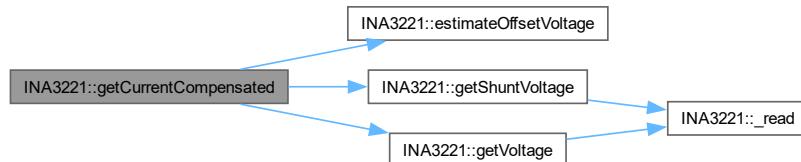


7.17.3.50 getCurrentCompensated()

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

Definition at line 246 of file [INA3221.cpp](#).

Here is the call graph for this function:

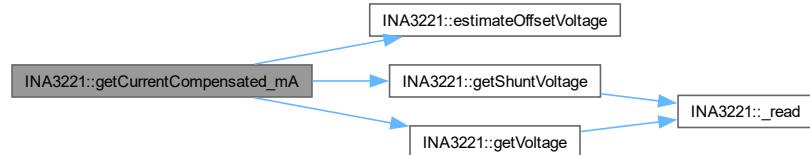


7.17.3.51 getCurrentCompensated_mA()

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

Definition at line 261 of file [INA3221.cpp](#).

Here is the call graph for this function:



7.17.3.52 getVoltage()

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

Definition at line 276 of file [INA3221.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



7.17.4 Member Data Documentation

7.17.4.1 _i2c

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

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

7.17.4.2 _i2c_addr

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

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

7.17.4.3 _shuntRes

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

Definition at line 114 of file [INA3221.h](#).

7.17.4.4 _filterRes

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

Definition at line 117 of file [INA3221.h](#).

7.17.4.5 _masken_reg

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

Definition at line 120 of file [INA3221.h](#).

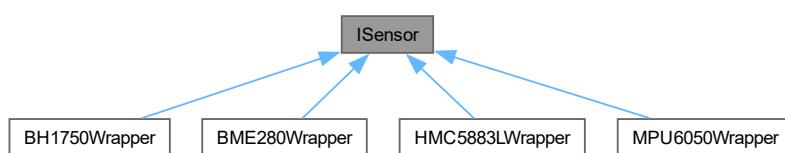
The documentation for this class was generated from the following files:

- 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 `readData (SensorDataTypeIdentifier type)=0`
- virtual bool `isInitialized () const =0`
- virtual `SensorType getType () 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 `readData()`

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

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

7.18.3.3 `isInitialized()`

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

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

7.18.3.4 `getType()`

```
virtual SensorType ISensor::getType () 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](#).

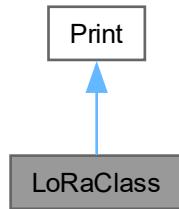
The documentation for this class was generated from the following file:

- lib/sensors/ISensor.h

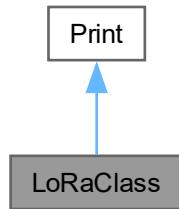
7.19 LoRaClass Class Reference

```
#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 parsePacket (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 setPins (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](#).

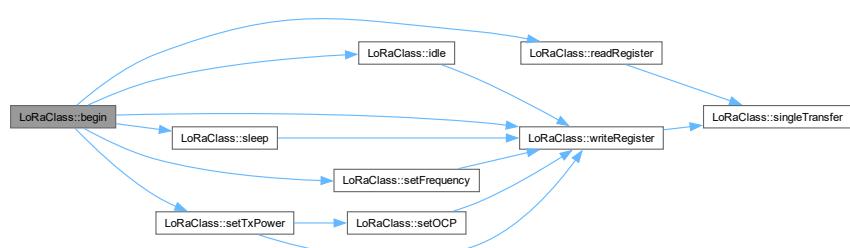
7.19.3 Member Function Documentation

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:

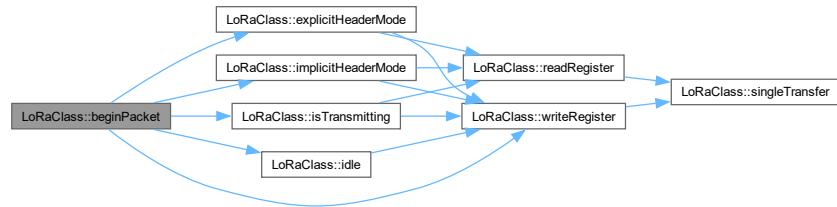


7.19.3.3 beginPacket()

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

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

Here is the call graph for this function:

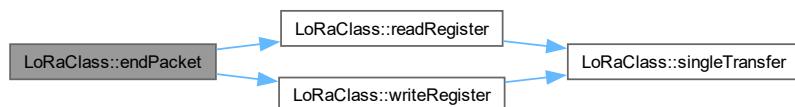


7.19.3.4 endPacket()

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

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

Here is the call graph for this function:

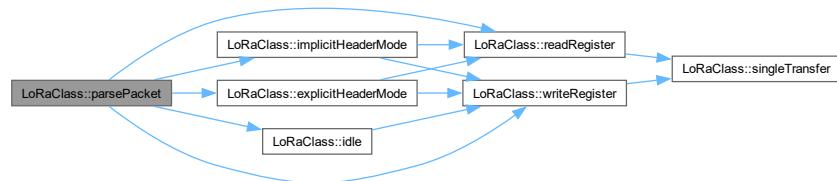


7.19.3.5 parsePacket()

```
int LoRaClass::parsePacket (
    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:



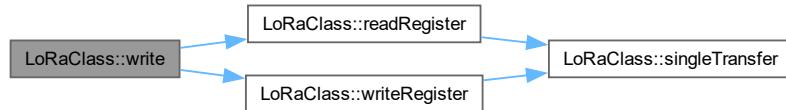
7.19.3.11 write() [2/2]

```
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()

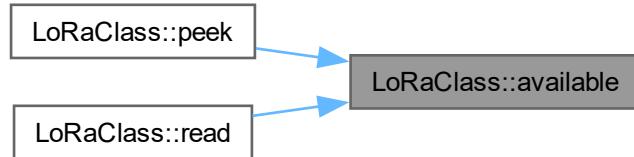
```
int LoRaClass::available ()  [virtual]
```

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:

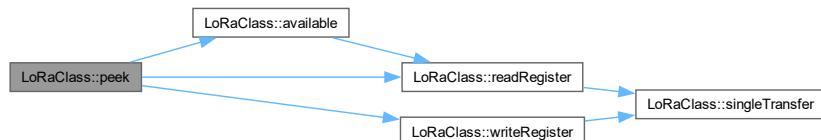


7.19.3.14 `peek()`

```
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:

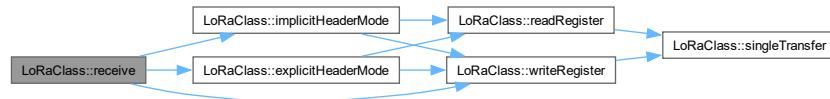


7.19.3.19 receive()

```
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()

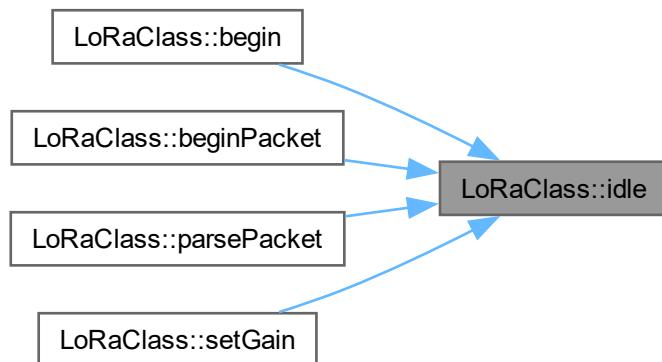
```
void LoRaClass::idle ()
```

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()

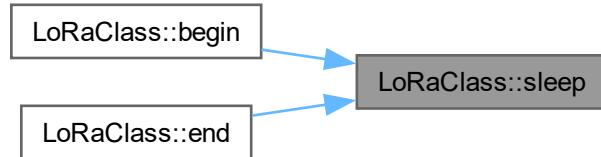
```
void LoRaClass::sleep ()
```

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:

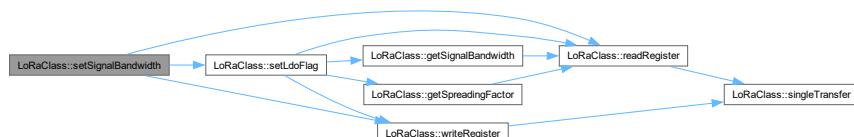


7.19.3.26 setSignalBandwidth()

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

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

Here is the call graph for this function:

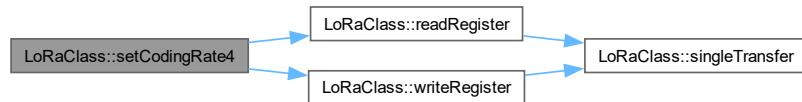


7.19.3.27 setCodingRate4()

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

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

Here is the call graph for this function:

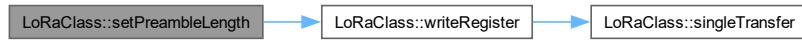


7.19.3.28 setPreambleLength()

```
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:

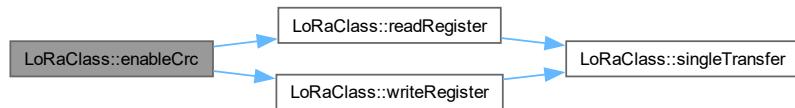


7.19.3.30 enableCrc()

```
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:

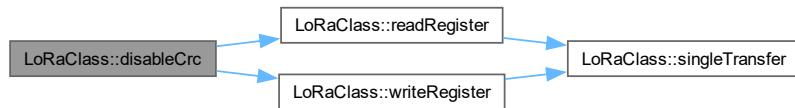


7.19.3.31 disableCrc()

```
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:

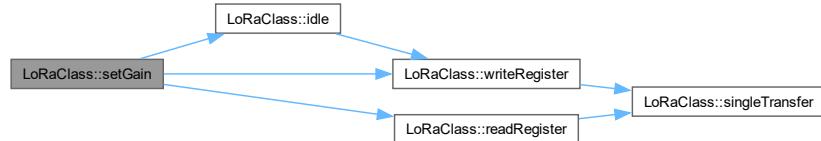


7.19.3.35 setGain()

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

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

Here is the call graph for this function:

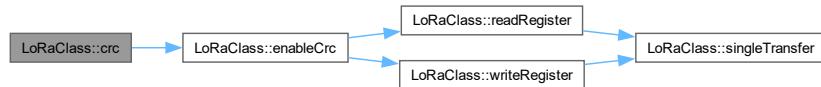


7.19.3.36 crc()

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

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

Here is the call graph for this function:

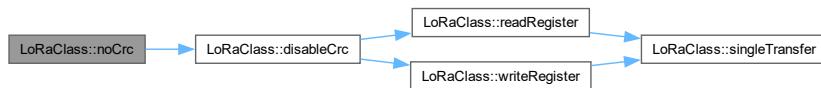


7.19.3.37 noCrc()

```
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 setPins()

```
void LoRaClass::setPins (
    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:

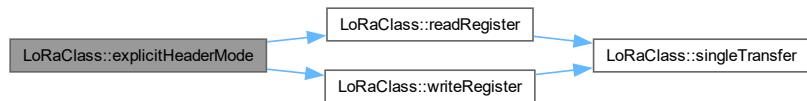


7.19.3.43 explicitHeaderMode()

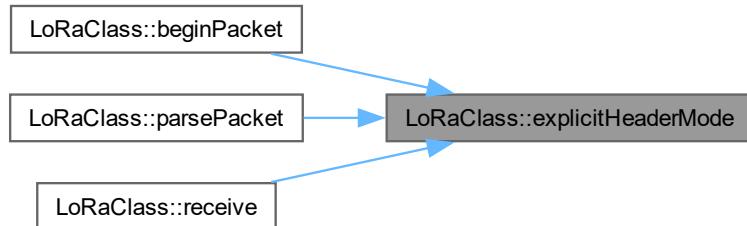
```
void LoRaClass::explicitHeaderMode () [private]
```

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:

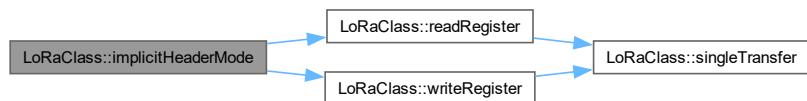


7.19.3.44 implicitHeaderMode()

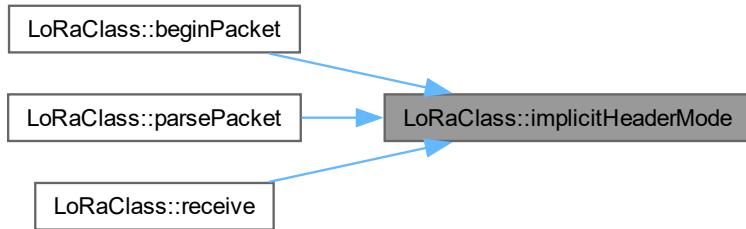
```
void LoRaClass::implicitHeaderMode () [private]
```

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:

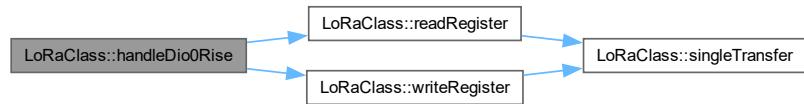


7.19.3.45 handleDio0Rise()

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

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

Here is the call graph for this function:

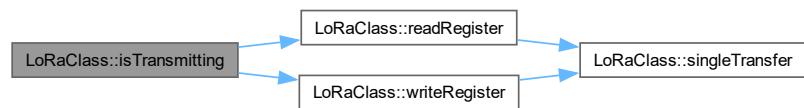


7.19.3.46 isTransmitting()

```
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:

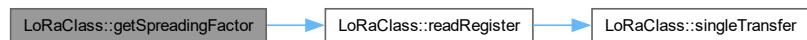


7.19.3.47 getSpreadingFactor()

```
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:

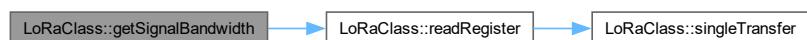


7.19.3.48 getSignalBandwidth()

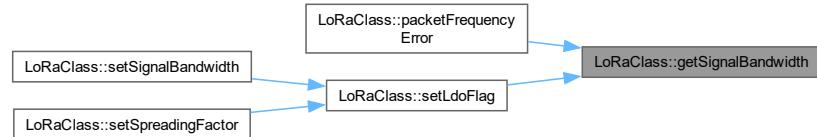
```
long LoRaClass::getSignalBandwidth () [private]
```

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:

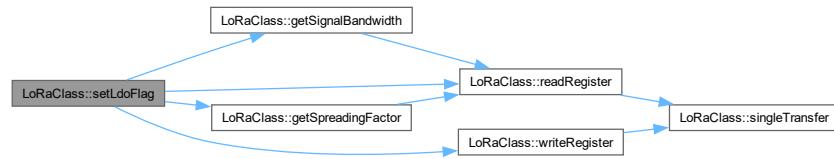


7.19.3.49 setLdoFlag()

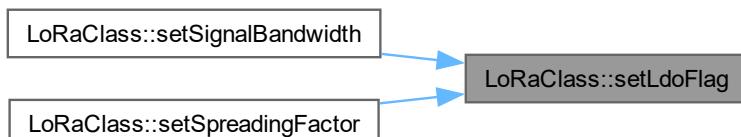
```
void LoRaClass::setLdoFlag () [private]
```

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()

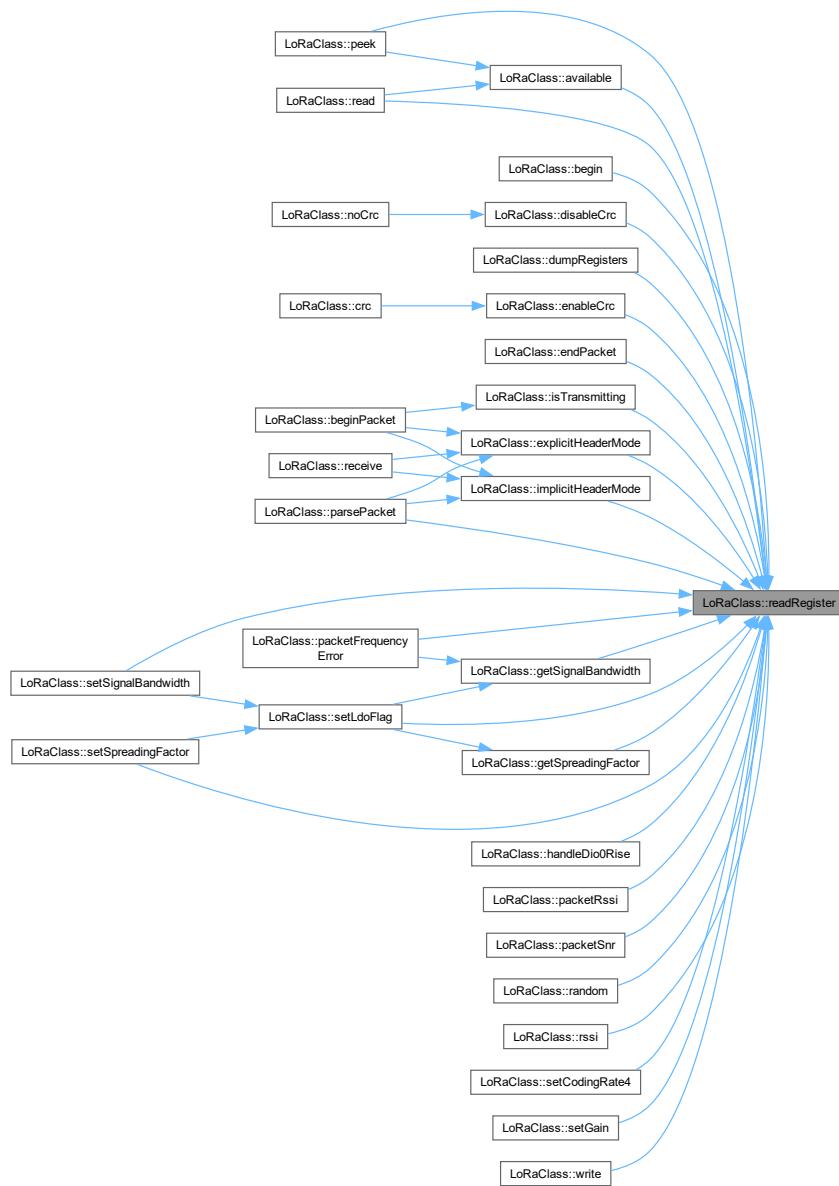
```
uint8_t LoRaClass::readRegister (
    uint8_t address) [private]
```

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()

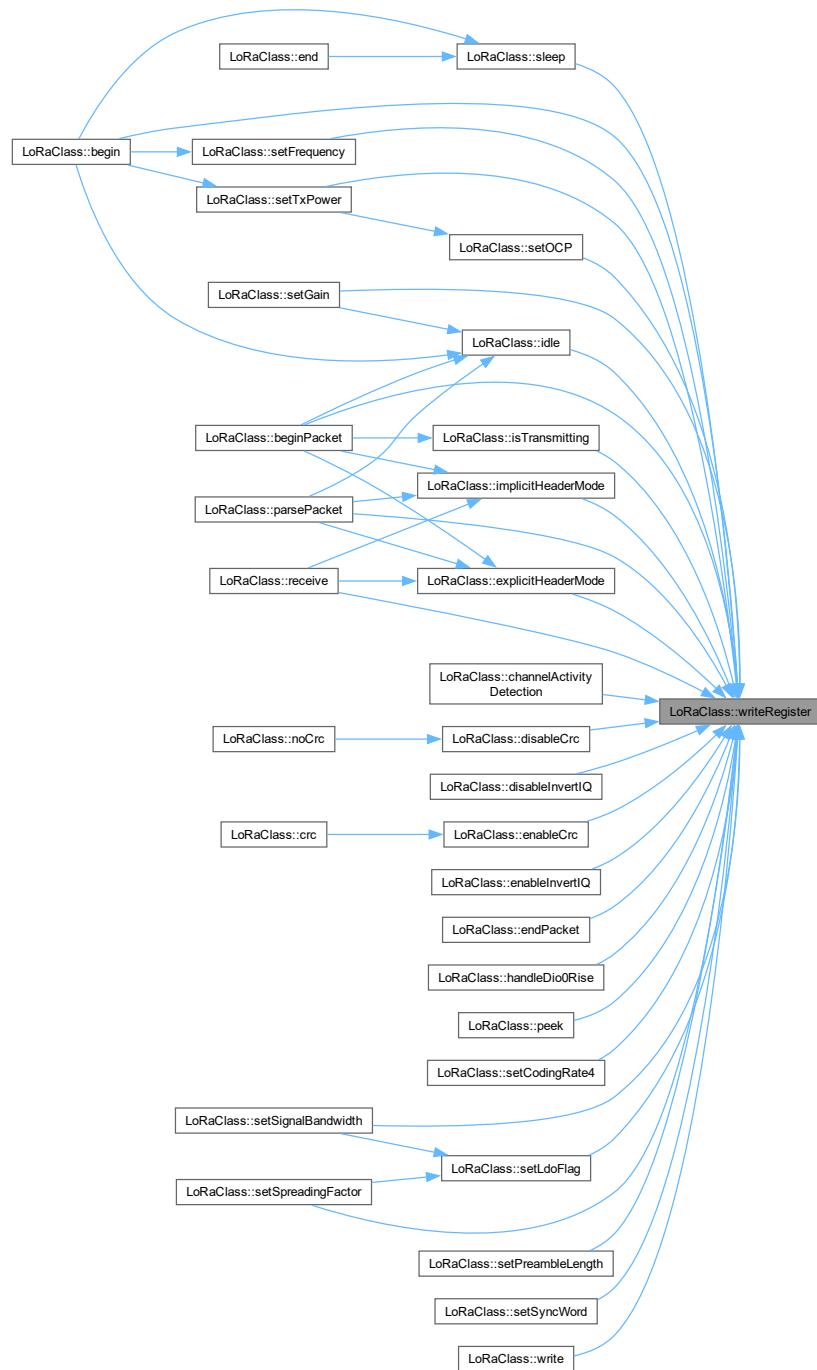
```
void LoRaClass::writeRegister (
    uint8_t address,
    uint8_t value) [private]
```

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:



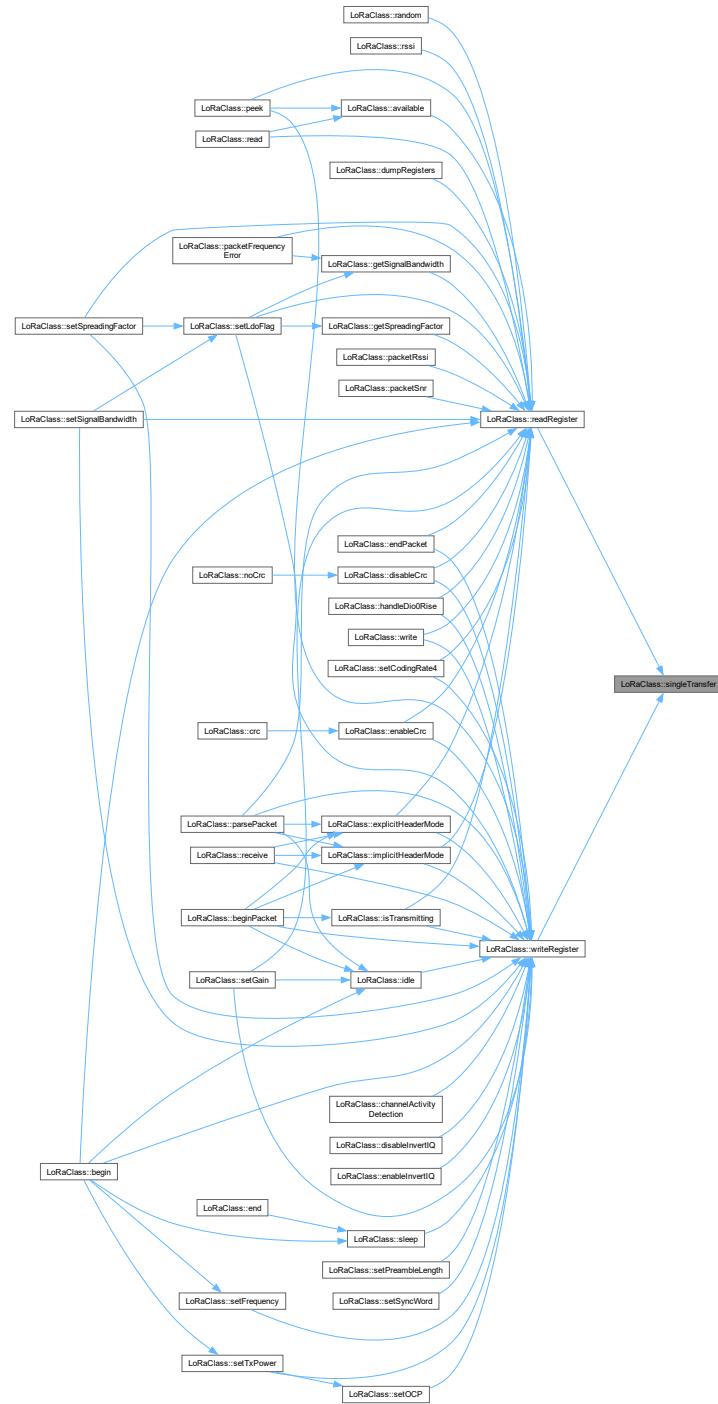
7.19.3.52 `singleTransfer()`

```

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:

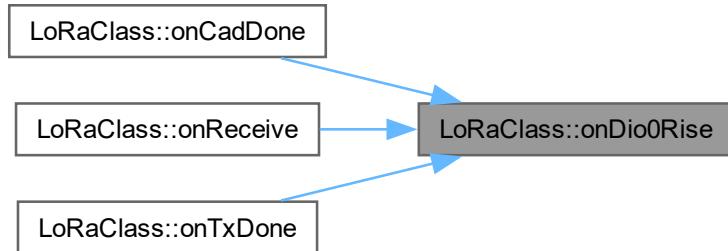


7.19.3.53 onDio0Rise()

```
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

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

Definition at line 90 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 91 of file [INA3221.h](#).

7.20.2.2 timing_ctrl_alert

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

Definition at line 92 of file [INA3221.h](#).

7.20.2.3 pwr_valid_alert

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

Definition at line 93 of file [INA3221.h](#).

7.20.2.4 warn_alert_ch3

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

Definition at line 94 of file [INA3221.h](#).

7.20.2.5 warn_alert_ch2

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

Definition at line 95 of file [INA3221.h](#).

7.20.2.6 warn_alert_ch1

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

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

7.20.2.7 shunt_sum_alert

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

Definition at line 97 of file [INA3221.h](#).

7.20.2.8 crit_alert_ch3

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

Definition at line 98 of file [INA3221.h](#).

7.20.2.9 crit_alert_ch2

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

Definition at line 99 of file [INA3221.h](#).

7.20.2.10 crit_alert_ch1

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

Definition at line 100 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 101 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 102 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 103 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 104 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 105 of file [INA3221.h](#).

7.20.2.16 reserved

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

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

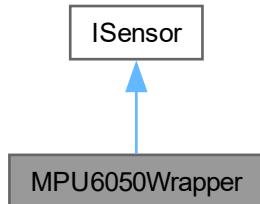
The documentation for this struct was generated from the following file:

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

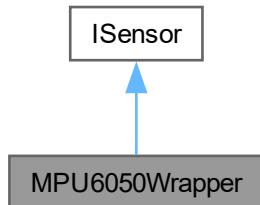
7.21 MPU6050Wrapper Class Reference

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

Inheritance diagram for MPU6050Wrapper:



Collaboration diagram for MPU6050Wrapper:



Public Member Functions

- `MPU6050Wrapper ()`
- `bool init () override`
- `float readData (SensorTypeIdentifier type) override`
- `bool isInitialized () const override`
- `SensorType getType () 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 `readData()`

```
float MPU6050Wrapper::readData (
    SensorTypeIdentifier type) [override], [virtual]
```

Implements [ISensor](#).

7.21.3.3 `isInitialized()`

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

Implements [ISensor](#).

7.21.3.4 `getType()`

```
SensorType MPU6050Wrapper::getType () 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 [updateRmcTokens](#) (const std::vector< std::string > &tokens)
- void [updateGgaTokens](#) (const std::vector< std::string > &tokens)
- std::vector< std::string > [getRmcTokens](#) () const
- std::vector< std::string > [getGgaTokens](#) () const

Private Attributes

- std::vector< std::string > `rmcTokens`
- std::vector< std::string > `ggaTokens`
- 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 updateRmcTokens()

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

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

7.22.3.2 updateGgaTokens()

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

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

7.22.3.3 getRmcTokens()

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

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

7.22.3.4 getGgaTokens()

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

Definition at line 29 of file [NMEA_data.cpp](#).

7.22.4 Member Data Documentation

7.22.4.1 rmcTokens

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

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

7.22.4.2 ggaTokens

```
std::vector<std::string> NMEAData::ggaTokens [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](#).

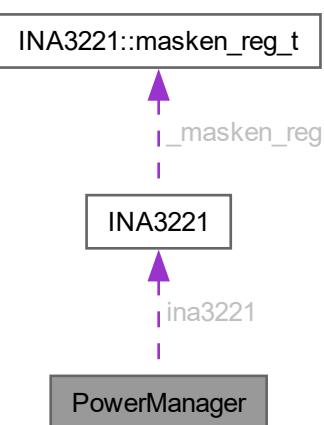
The documentation for this class was generated from the following files:

- 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 readIDs ()`
- `float getCurrentChargeSolar ()`
- `float getCurrentChargeUSB ()`
- `float getCurrentChargeTotal ()`
- `float getCurrentDraw ()`
- `float getVoltageBattery ()`
- `float getVoltage5V ()`
- `void configure (const std::map< std::string, std::string > &config)`
- `bool isSolarActive ()`
- `bool isUSBConnected ()`

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`
- `bool solarActive = false`
- `bool usbConnected = 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 4 of file [PowerManager.cpp](#).

7.23.3 Member Function Documentation

7.23.3.1 initialize()

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

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

7.23.3.2 `readIDs()`

```
std::string PowerManager::readIDs ()
```

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

7.23.3.3 `getCurrentChargeSolar()`

```
float PowerManager::getCurrentChargeSolar ()
```

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

Here is the caller graph for this function:



7.23.3.4 `getCurrentChargeUSB()`

```
float PowerManager::getCurrentChargeUSB ()
```

Definition at line 29 of file [PowerManager.cpp](#).

Here is the caller graph for this function:



7.23.3.5 `getCurrentChargeTotal()`

```
float PowerManager::getCurrentChargeTotal ()
```

Definition at line 44 of file [PowerManager.cpp](#).

7.23.3.6 `getCurrentDraw()`

```
float PowerManager::getCurrentDraw ()
```

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

7.23.3.7 `getVoltageBattery()`

```
float PowerManager::getVoltageBattery ()
```

Definition at line 19 of file [PowerManager.cpp](#).

7.23.3.8 `getVoltage5V()`

```
float PowerManager::getVoltage5V ()
```

Definition at line 24 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)
```

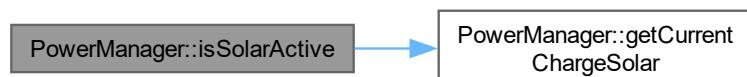
Definition at line 49 of file [PowerManager.cpp](#).

7.23.3.10 `isSolarActive()`

```
bool PowerManager::isSolarActive ()
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



7.23.3.11 `isUSBConnected()`

```
bool PowerManager::isUSBConnected ()
```

Definition at line 81 of file [PowerManager.cpp](#).

Here is the call graph for this function:



Here is the caller 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 26 of file [PowerManager.h](#).

7.23.4.2 `USB_CURRENT_THRESHOLD`

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

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

7.23.4.3 `VOLTAGE_LOW_THRESHOLD`

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

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

7.23.4.4 VOLTAGE_OVERCHARGE_THRESHOLD

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

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

7.23.4.5 FALL_RATE_THRESHOLD

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

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

7.23.4.6 FALLING_TREND_REQUIRED

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

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

7.23.4.7 ina3221

```
INA3221 PowerManager::ina3221 [private]
```

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

7.23.4.8 initialized

```
bool PowerManager::initialized [private]
```

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

7.23.4.9 solarActive

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

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

7.23.4.10 usbConnected

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

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

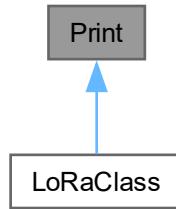
The documentation for this class was generated from the following files:

- 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()

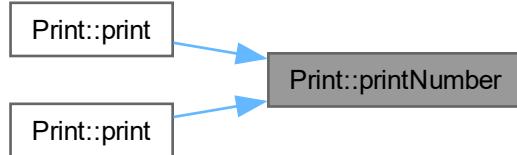
```
size_t Print::printNumber (
    unsigned long n,
    uint8_t base) [private]
```

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()

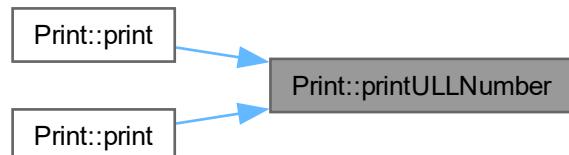
```
size_t Print::printULLNumber (
    unsigned long long n64,
    uint8_t base) [private]
```

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()

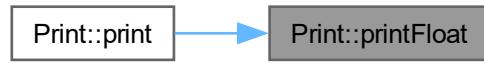
```
size_t Print::printFloat (
    double number,
    int digits) [private]
```

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:

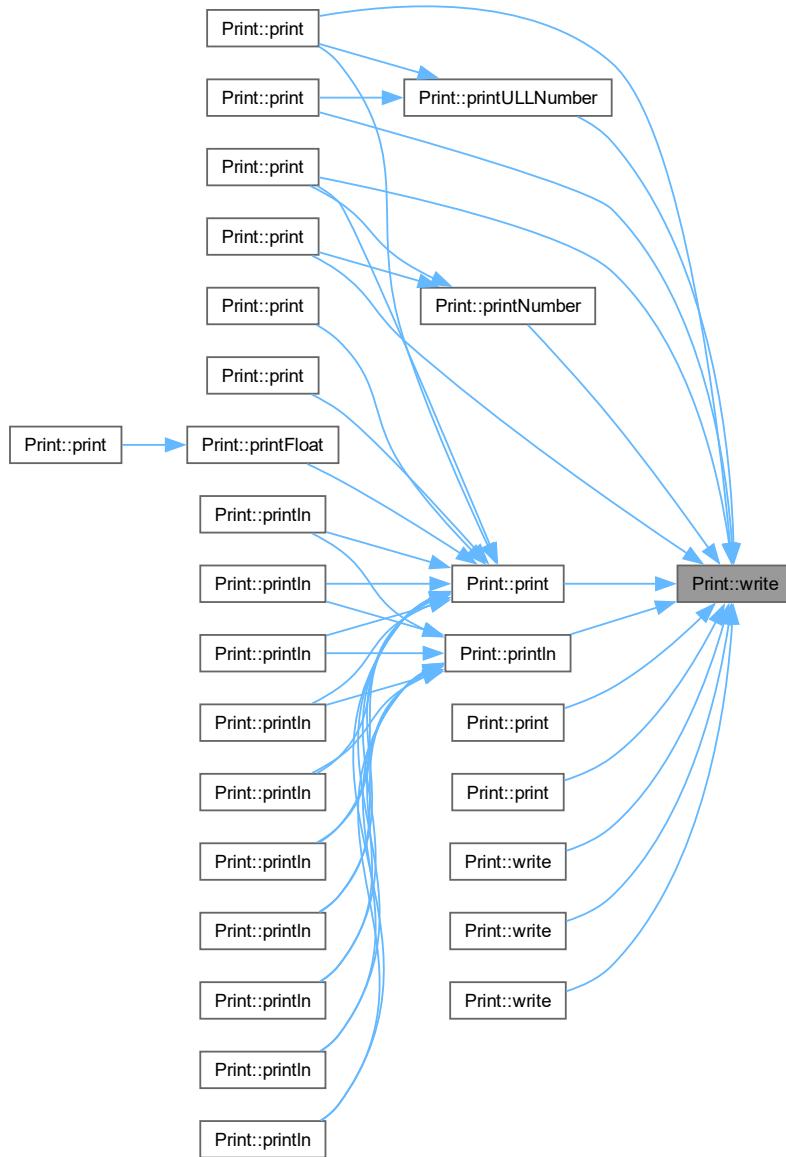


7.24.3.7 write() [1/4]

```
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]

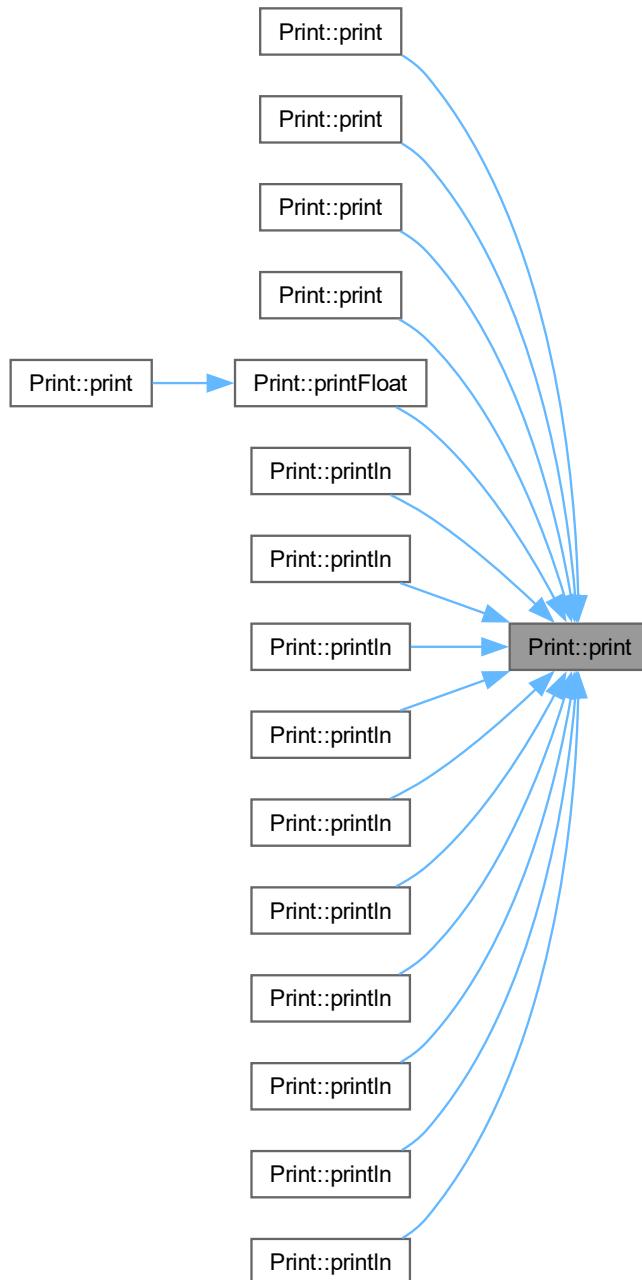
```
size_t Print::print (
    char c)
```

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:

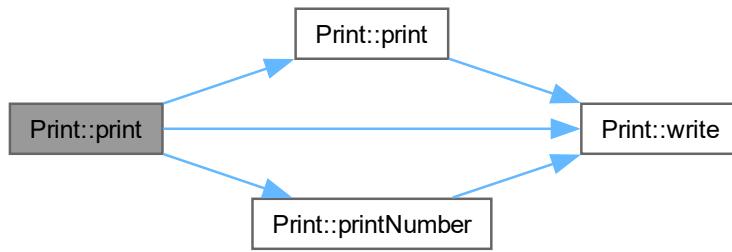


7.24.3.18 print() [7/11]

```
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:

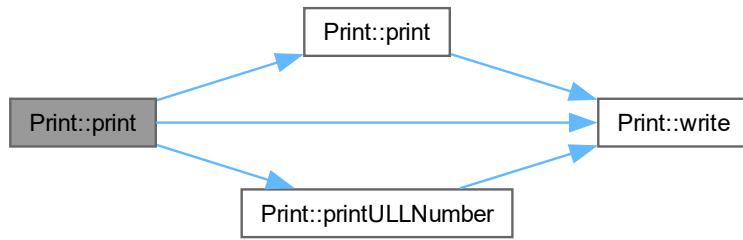


7.24.3.20 print() [9/11]

```
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](#).

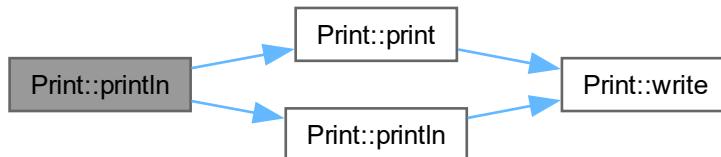
Here is the call graph for this function:

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

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

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

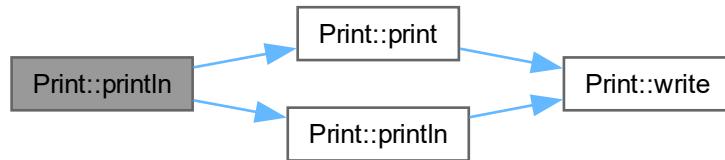
Here is the call graph for this function:

**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:

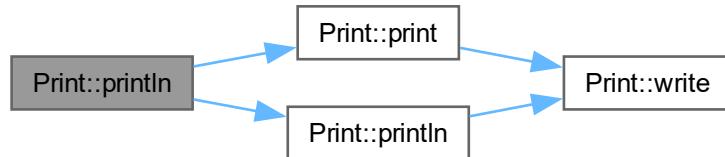


7.24.3.25 println() [3/11]

```
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:

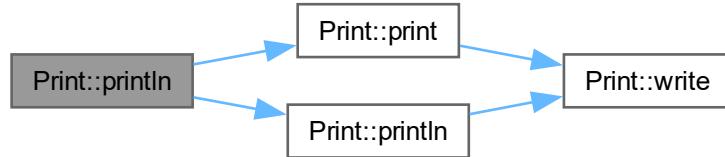


7.24.3.26 println() [4/11]

```
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:

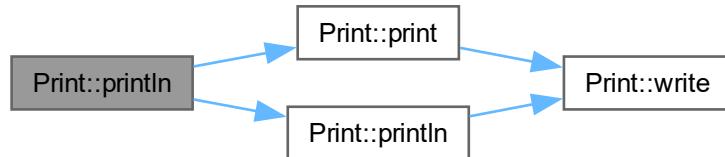


7.24.3.27 `println()` [5/11]

```
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:

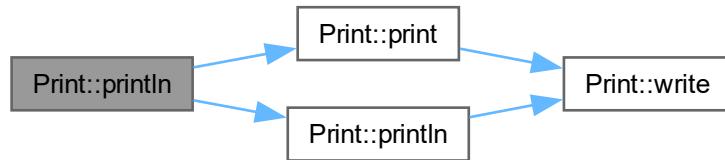


7.24.3.28 `println()` [6/11]

```
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:

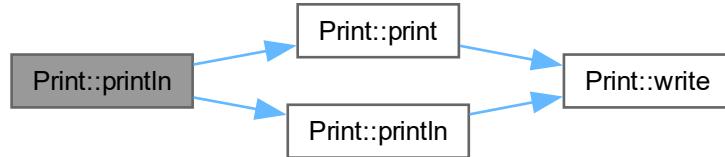


7.24.3.29 println() [7/11]

```
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:

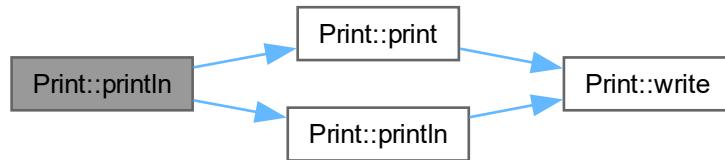


7.24.3.30 println() [8/11]

```
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:

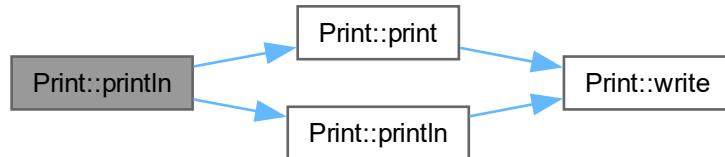


7.24.3.31 `println()` [9/11]

```
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:

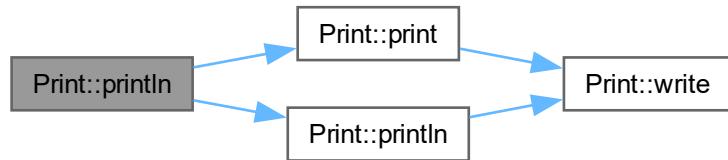


7.24.3.32 `println()` [10/11]

```
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]

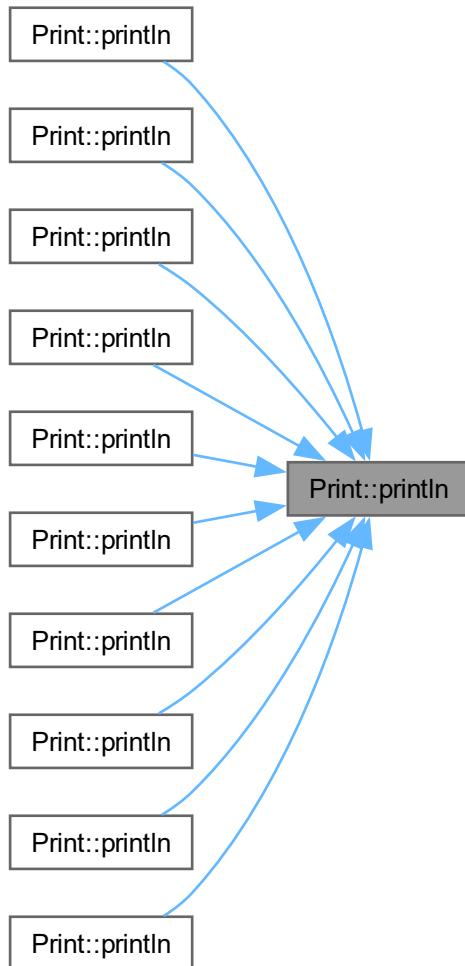
```
size_t Print::println (
    void )
```

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 [initSensor \(SensorType type, i2c_inst_t *i2c=nullptr\)](#)
Initializes a given sensor type on the specified I2C bus.
- bool [configureSensor \(SensorType type, const std::map< std::string, std::string > &config\)](#)
Configures an already initialized sensor with supplied settings.
- float [readSensorData \(SensorType sensorType, SensorDataTypelIdentifier dataType\)](#)
Reads a specific data type (e.g., temperature, humidity) from a sensor.

Static Public Member Functions

- static [SensorWrapper & getInstance \(\)](#)
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 getInstance()

```
SensorWrapper & SensorWrapper::getInstance () [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 initSensor()

```
bool SensorWrapper::initSensor (
    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 configureSensor()

```
bool SensorWrapper::configureSensor (
    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 readSensorData()

```
float SensorWrapper::readSensorData (
    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:

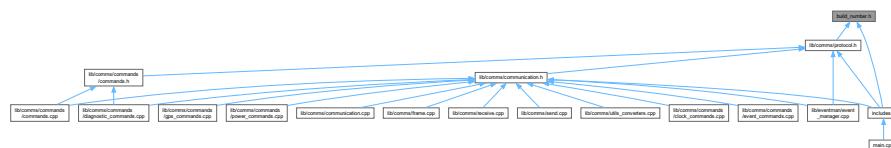
- lib/sensors/[ISensor.h](#)
- lib/sensors/[ISensor.cpp](#)

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 3`

8.1.1 Macro Definition Documentation

8.1.1.1 BUILD_NUMBER

```
#define BUILD_NUMBER 3
```

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 3
00007
00008 #endif
```

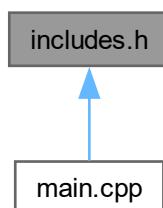
8.3 includes.h File Reference

```
#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.](#)

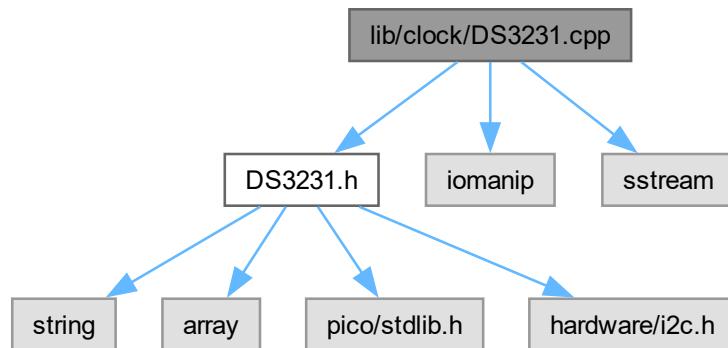
```

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 <iomanip>
#include <sstream>
Include dependency graph for DS3231.cpp:
```



8.6 DS3231.cpp

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

```

00001 #include "DS3231.h"
00002 #include <iomanip>
00003 #include <sstream>
00004
00005 const std::array<std::string, 7> DS3231::WEEKDAYS = {
00006     "SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"
00007 };
00008
00009 DS3231::DS3231(i2c_inst_t *i2c_port, uint8_t address) : i2c(i2c_port), address(address) {
00010 }
00011
00012 bool DS3231::setTime(uint8_t sec, uint8_t min, uint8_t hour,
00013                         uint8_t weekday, uint8_t day, uint8_t month, uint16_t year) {
00014     uint8_t buffer[8];
00015     buffer[0] = 0;
00016     buffer[1] = bin2bcd(sec);
00017     buffer[2] = bin2bcd(min);
00018     buffer[3] = bin2bcd(hour);
00019     buffer[4] = bin2bcd(weekday);
00020     buffer[5] = bin2bcd(day);
00021     buffer[6] = bin2bcd(month);
00022     buffer[7] = bin2bcd(year - 2000);
00023     // Write time to RTC
00024     uint8_t reg = RTC_REGISTER;
00025     i2c_write_blocking(i2c, address, &reg, 1, true);
00026     return i2c_write_blocking(i2c, address, buffer, sizeof(buffer), false) == sizeof(buffer);
00027 }
00028
00029 bool DS3231::getTime(uint8_t& sec, uint8_t& min, uint8_t& hour,
00030                         std::string& weekday, uint8_t& day, uint8_t& month, uint16_t& year) {
00031     uint8_t buffer[7];
00032     uint8_t reg = RTC_REGISTER;
00033
00034     if (i2c_write_blocking(i2c, address, &reg, 1, true) != 1) {
00035         return false;
00036     }
00037
00038     if (i2c_read_blocking(i2c, address, buffer, sizeof(buffer), false) != sizeof(buffer)) {
00039         return false;
00040     }
00041
00042     sec = bcd2bin(buffer[0]);
00043     min = bcd2bin(buffer[1]);
00044     hour = bcd2bin(buffer[2]);
00045     weekday = WEEKDAYS[bcd2bin(buffer[3])];
00046     day = bcd2bin(buffer[4]);
00047     month = bcd2bin(buffer[5]);
00048     year = bcd2bin(buffer[6]) + 2000;
00049
00050     return true;
00051 }
00052
00053 std::string DS3231::getTimeString() {
00054     uint8_t sec, min, hour, day, month;
00055     uint16_t year;
00056     std::string weekday;
00057
00058     if (!getTime(sec, min, hour, weekday, day, month, year)) {
00059         return "Error reading RTC";
00060     }
00061
00062     std::stringstream ss;
00063     ss << preZero(hour) << ":" << preZero(min) << ":" << preZero(sec)
00064     << " " << weekday << " "
00065     << static_cast<int>(day) << "."
00066     << static_cast<int>(month) << "."
00067     << year;
00068
00069     return ss.str();
00070 }
00071
00072 uint8_t DS3231::bcd2bin(uint8_t val) {
00073     return ((val/16) * 10) + (val % 16);
00074 }
00075
00076 uint8_t DS3231::bin2bcd(uint8_t val) {
00077     return ((val/10) * 16) + (val % 10);
00078 }
00079
00080 std::string DS3231::preZero(uint8_t val) {
00081     if (val < 10) {
00082         return "0" + std::to_string(val);

```

```

00083     }
00084     return std::to_string(val);
00085 }
00086
00087 bool DS3231::setTimeUnix(uint32_t unixTime) {
00088     DateTime dt = unixToDateTIme(unixTime);
00089     uint8_t weekdayIndex = 0;
00090     // Calculate weekday (0 = Sunday, 1 = Monday, etc.)
00091     time_t t = unixTime;
00092     struct tm* tmp = gmtime(&t);
00093     weekdayIndex = tmp->tm_wday;
00094
00095     return setTime(dt.second, dt.minute, dt.hour,
00096                     weekdayIndex, dt.day, dt.month, dt.year);
00097 }
00098
00099 uint32_t DS3231::getTimeUnix() {
00100     DateTime dt = getDateTIme();
00101     return dateTImeToUnix(dt);
00102 }
00103
00104 DateTime DS3231::getDateTIme() {
00105     DateTime dt;
00106     uint8_t buffer[7];
00107     uint8_t reg = RTC_REGISTER;
00108
00109     if (i2c_write_blocking(i2c, address, &reg, 1, true) != 1) {
00110         return Datetime{0};
00111     }
00112
00113     if (i2c_read_blocking(i2c, address, buffer, sizeof(buffer), false) != sizeof(buffer)) {
00114         return DateTime{0};
00115     }
00116
00117     dt.second = bcd2bin(buffer[0]);
00118     dt.minute = bcd2bin(buffer[1]);
00119     dt.hour = bcd2bin(buffer[2]);
00120     dt.weekday = WEEKDAYS[bcd2bin(buffer[3])];
00121     dt.day = bcd2bin(buffer[4]);
00122     dt.month = bcd2bin(buffer[5]);
00123     dt.year = bcd2bin(buffer[6]) + 2000;
00124
00125     return dt;
00126 }
00127
00128 uint64_t DS3231::getTimeInteger() {
00129     DateTime dt = getDateTIme();
00130     return static_cast<uint64_t>(dt.year) * 10000000000ULL +
00131             static_cast<uint64_t>(dt.month) * 100000000ULL +
00132             static_cast<uint64_t>(dt.day) * 100000ULL +
00133             static_cast<uint64_t>(dt.hour) * 10000ULL +
00134             static_cast<uint64_t>(dt.minute) * 100ULL +
00135             static_cast<uint64_t>(dt.second);
00136 }
00137
00138 uint32_t DS3231::dateTImeToUnix(const DateTime& dt) {
00139     struct tm timeinfo = {};
00140     timeinfo.tm_year = dt.year - 1900;
00141     timeinfo.tm_mon = dt.month - 1;
00142     timeinfo.tm_mday = dt.day;
00143     timeinfo.tm_hour = dt.hour;
00144     timeinfo.tm_min = dt.minute;
00145     timeinfo.tm_sec = dt.second;
00146     return mktime(&timeinfo);
00147 }
00148
00149 DateTime DS3231::unixToDateTIme(uint32_t unixTime) {
00150     DateTime dt;
00151     time_t t = unixTime;
00152     struct tm* tmp = gmtime(&t);
00153
00154     dt.year = tmp->tm_year + 1900;
00155     dt.month = tmp->tm_mon + 1;
00156     dt.day = tmp->tm_mday;
00157     dt.hour = tmp->tm_hour;
00158     dt.minute = tmp->tm_min;
00159     dt.second = tmp->tm_sec;
00160     dt.weekday = WEEKDAYS[tmp->tm_wday];
00161
00162     return dt;
00163 }
00164
00165 uint32_t DS3231::getTimeUnixLocal() {
00166     uint32_t utcTime = getTimeUnix();
00167     return utcTime + (timezoneOffset * 60); // Convert minutes to seconds
00168 }
00169

```

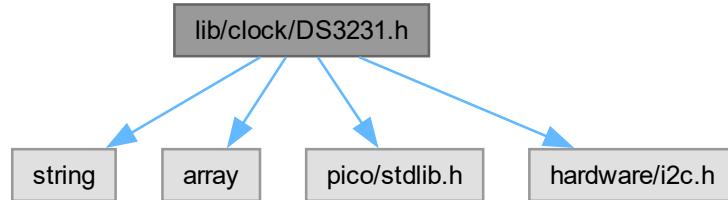
```

00170 DateTime DS3231::getDateTimeLocal() {
00171     DateTime utc = getDateTime();
00172     return applyTimezone(utc, timezoneOffset);
00173 }
00174
00175 DateTime DS3231::applyTimezone(const DateTime& utc, int16_t offsetMinutes) {
00176     time_t t = dateTimeToUnix(utc) + (offsetMinutes * 60);
00177     return unixToDate(t);
00178 }

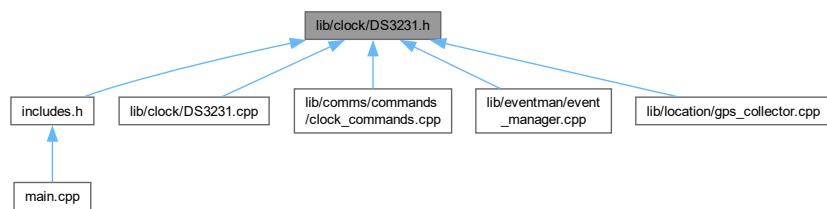
```

8.7 lib/clock/DS3231.h File Reference

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



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



Classes

- struct `DateTime`
- class `DS3231`

8.8 DS3231.h

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

```

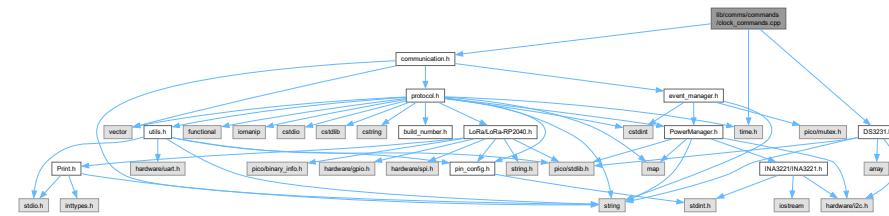
00001 #ifndef DS3231_H
00002 #define DS3231_H
00003
00004 #include <string>
00005 #include <array>
00006 #include "pico/stl.h"
00007 #include "hardware/i2c.h"
00008
00009 struct DateTime {
00010     uint16_t year;
00011     uint8_t month;
00012     uint8_t day;
00013     uint8_t hour;
00014     uint8_t minute;
00015     uint8_t second;
00016     std::string weekday;
00017 };
00018
00019 class DS3231 {
00020 public:
00021     // Weekday names
00022     static const std::array<std::string, 7> WEEKDAYS;
00023
00024     // Constructor
00025     DS3231(i2c_inst_t *i2c, uint8_t address = 0x68);
00026
00027     // Set time using binary-coded decimal format
00028     bool setTime(uint8_t sec, uint8_t min, uint8_t hour,
00029                  uint8_t weekday, uint8_t day, uint8_t month, uint16_t year);
00030
00031     // Get time in two formats
00032     bool getTime(uint8_t& sec, uint8_t& min, uint8_t& hour,
00033                  std::string& weekday, uint8_t& day, uint8_t& month, uint16_t& year);
00034     std::string getTimeString();
00035     bool setTimeUnix(uint32_t unixTime);
00036     uint32_t getTimeUnix();
00037     DateTime getDate();
00038     uint64_t getTimeInteger(); // Returns YYYYMMDDHHMMSS
00039
00040     void setTimezoneOffset(int16_t offsetMinutes) { timezoneOffset = offsetMinutes; }
00041     int16_t getTimezoneOffset() const { return timezoneOffset; }
00042
00043     uint32_t getTimeUnixLocal();
00044     DateTime getDateLocal();
00045
00046     void setClockSyncInterval(uint32_t intervalSeconds) { syncInterval = intervalSeconds; }
00047     uint32_t getClockSyncInterval() const { return syncInterval; }
00048
00049     void setLastSyncTime(uint32_t unixTime) { lastSyncTime = unixTime; }
00050     uint32_t getLastSyncTime() const { return lastSyncTime; }
00051
00052
00053 private:
00054     i2c_inst_t* i2c;
00055     uint8_t address;
00056     static constexpr uint8_t RTC_REGISTER = 0x00;
00057
00058     int16_t timezoneOffset = 0;      // Offset in minutes from UTC
00059     uint32_t syncInterval = 86400;    // Default sync interval: 24 hours
00060     uint32_t lastSyncTime = 0;       // Last successful GPS sync time
00061
00062     // Utility functions
00063     uint8_t bcd2bin(uint8_t val);
00064     uint8_t bin2bcd(uint8_t val);
00065     std::string preZero(uint8_t val);
00066     static uint32_t dateTimeToUnix(const DateTime& dt);
00067     static DateTime unixToDate(uint32_t unixTime);
00068     DateTime applyTimezone(const DateTime& utc, int16_t offsetMinutes);
00069
00070 };
00071
00072 #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:
```



Functions

- **Frame handleTime (const std::string ¶m, OperationType operationType)**
Handler for getting and setting system time.
- **Frame handleTimezoneOffset (const std::string ¶m, OperationType operationType)**
Handler for getting and setting timezone offset.
- **Frame handleClockSyncInterval (const std::string ¶m, OperationType operationType)**
Handler for getting and setting clock synchronization interval.
- **Frame handleGetLastSyncTime (const std::string ¶m, OperationType operationType)**
Handler for getting last clock sync time.

Variables

- **DS3231 systemClock**

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 // Declare the systemClock as extern
00006 extern DS3231 systemClock;
00007
00008
00009
00010 extern DS3231 systemClock;
00011
00012
00013
00014
00015
00016
00017
00018
00019
00020
00021
00022
00023
00024
00025
00026 Frame handleTime(const std::string& param, OperationType operationType) {
00027     if (operationType == OperationType::SET && param.empty()) {
00028         return buildFrame(ExecutionResult::ERROR, 3, 0, "PARAM REQUIRED");
00029     }
00030
00031     if (operationType == OperationType::GET && !param.empty()) {
00032         return buildFrame(ExecutionResult::ERROR, 3, 0, "PARAM UNNECESSARY");
00033     }
00034
00035     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00036         return buildFrame(ExecutionResult::ERROR, 3, 0, "INVALID OPERATION");
00037     }
00038
00039     if (operationType == OperationType::SET) {
00040         try {
00041             time_t newTime = std::stoll(param);
00042             struct tm* timeinfo = localtime(&newTime);
00043
00044             if (timeinfo != nullptr) {
00045                 // Set the time on the RTC
00046
00047                 if (settimeofday(timeinfo, NULL) < 0) {
00048                     return buildFrame(ExecutionResult::ERROR, 3, 0, "TIME SETTING FAILED");
00049                 }
00050             }
00051         } catch (...) {
00052             return buildFrame(ExecutionResult::ERROR, 3, 0, "TIME CONVERSION FAILED");
00053         }
00054     }
00055
00056     return buildFrame(ExecutionResult::OK, 3, 0, "TIME SETTING SUCCESSFUL");
00057 }
```

```

00046             systemClock.setTime(timeinfo->tm_sec, timeinfo->tm_min, timeinfo->tm_hour,
00047                                     timeinfo->tm_wday, timeinfo->tm_mday, timeinfo->tm_mon + 1,
00048                                     timeinfo->tm_year + 1900);
00049             EventEmitter::emit(EventGroup::CLOCK, ClockEvent::CHANGED);
00050         } else {
00051             return buildFrame(ExecutionResult::SUCCESS, 3, 0, "Time set successfully");
00052         }
00053     } catch (...) {
00054         return buildFrame(ExecutionResult::ERROR, 3, 0, "FAILED TO SET TIME");
00055     }
00056 }
00057
00058 // GET operation
00059 uint8_t sec, min, hour, day, month;
00060 uint16_t year;
00061 std::string weekday;
00062 if (systemClock.getTime(sec, min, hour, weekday, day, month, year)) {
00063     std::stringstream ss;
00064     ss << std::setw(2) << std::setfill('0') << static_cast<int>(hour) << ":" 
00065     << std::setw(2) << std::setfill('0') << static_cast<int>(min) << ":" 
00066     << std::setw(2) << std::setfill('0') << static_cast<int>(sec) << " "
00067     << weekday << " "
00068     << std::setw(2) << std::setfill('0') << static_cast<int>(day) << "."
00069     << std::setw(2) << std::setfill('0') << static_cast<int>(month) << "."
00070     << year;
00071     return buildFrame(ExecutionResult::SUCCESS, 3, 0, ss.str());
00072 } else {
00073     return buildFrame(ExecutionResult::ERROR, 3, 0, "FAILED TO GET TIME");
00074 }
00075 }
00076
00077
00078 Frame handleTimezoneOffset(const std::string& param, OperationType operationType) {
00079     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00080         return buildFrame(ExecutionResult::ERROR, 3, 1, "INVALID OPERATION");
00081     }
00082
00083     if (operationType == OperationType::GET) {
00084         if (!param.empty()) {
00085             return buildFrame(ExecutionResult::ERROR, 3, 1, "PARAM UNNECESSARY");
00086         }
00087         return buildFrame(ExecutionResult::SUCCESS, 3, 1,
00088                           std::to_string(systemClock.getTimezoneOffset()));
00089     }
00090
00091     if (operationType == OperationType::SET) {
00092         if (param.empty()) {
00093             return buildFrame(ExecutionResult::ERROR, 3, 1, "PARAM REQUIRED");
00094         }
00095         try {
00096             int16_t offset = std::stoi(param);
00097             if (offset < -720 || offset > 720) { // ±12 hours in minutes
00098                 return buildFrame(ExecutionResult::ERROR, 3, 1, "INVALID OFFSET");
00099             }
00100             systemClock.setTimezoneOffset(offset);
00101             return buildFrame(ExecutionResult::SUCCESS, 3, 1, param);
00102         } catch (...) {
00103             return buildFrame(ExecutionResult::ERROR, 3, 1, "INVALID PARAMETER");
00104         }
00105     }
00106     return buildFrame(ExecutionResult::ERROR, 3, 1, "UNKNOWN ERROR");
00107 }
00108
00109
00110 Frame handleClockSyncInterval(const std::string& param, OperationType operationType) {
00111     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00112         return buildFrame(ExecutionResult::ERROR, 3, 2, "INVALID OPERATION");
00113     }
00114
00115     if (operationType == OperationType::GET) {
00116         if (!param.empty()) {
00117             return buildFrame(ExecutionResult::ERROR, 3, 2, "PARAM UNNECESSARY");
00118         }
00119         return buildFrame(ExecutionResult::SUCCESS, 3, 2,
00120                           std::to_string(systemClock.getClockSyncInterval()));
00121     }
00122
00123     if (operationType == OperationType::SET) {
00124         if (param.empty()) {
00125             return buildFrame(ExecutionResult::ERROR, 3, 2, "PARAM REQUIRED");
00126         }
00127         try {
00128             uint32_t interval = std::stoul(param);
00129             systemClock.setClockSyncInterval(interval);
00130             return buildFrame(ExecutionResult::SUCCESS, 3, 2, param);
00131         } catch (...) {
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151 }
```

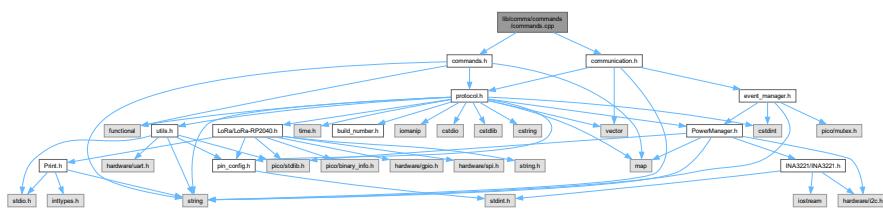
```

00152         return buildFrame(ExecutionResult::ERROR, 3, 2, "INVALID PARAMETER");
00153     }
00154 }
00155 return buildFrame(ExecutionResult::ERROR, 3, 2, "UNKNOWN ERROR");
00156 }
00157
00167 Frame handleGetLastSyncTime(const std::string& param, OperationType operationType) {
00168     if (operationType != OperationType::GET || !param.empty()) {
00169         return buildFrame(ExecutionResult::ERROR, 3, 3, "INVALID REQUEST");
00170     }
00171     return buildFrame(ExecutionResult::SUCCESS, 3, 3,
00172                         std::to_string(systemClock.getLastSyncTime()));
00173 } // 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 executeCommand (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)), handleListCommands},           //
00029     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(1)), handleGetBuildVersion},          //
00030     {((static_cast<uint32_t>(1) << 8) | static_cast<uint32_t>(9)), handleEnterBootloaderMode},        //
00031     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(0)), handleGetPowerManagerIDs},         //
00032     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(2)), handleGetVoltageBattery},          //
00033     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(3)), handleGetVoltage5V},            //
00034     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(4)), handleGetCurrentChargeUSB},       //
00035     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(5)), handleGetCurrentChargeSolar},      //
00036     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(6)), handleGetCurrentChargeTotal},      //
00037     {((static_cast<uint32_t>(2) << 8) | static_cast<uint32_t>(7)), handleGetCurrentDraw},            //
00038     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(0)), handleTime},                      //
00039     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(1)), handleTimezoneOffset},           // Group 3,
00040     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(2)), handleClockSyncInterval},        // Group
00041     {((static_cast<uint32_t>(3) << 8) | static_cast<uint32_t>(3)), handleGetLastSyncTime},          //
00042     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(1)), handleGetLastEvents},           // Group 5,
00043     {((static_cast<uint32_t>(5) << 8) | static_cast<uint32_t>(2)), handleGetEventCount},           // Group 5,
00044     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(1)), handleGPSPowerStatus},        //
00045     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(2)), handleEnableGPSTransparentMode}, //
00046     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(3)), handleGetRMCData},             //
00047     {((static_cast<uint32_t>(7) << 8) | static_cast<uint32_t>(4)), handleGetGGAData},           //
00048 };
00049
00050
00059 Frame executeCommand(uint32_t commandKey, const std::string& param, OperationType operationType) {
00060     auto it = commandHandlers.find(commandKey);
00061     if (it != commandHandlers.end()) {
00062         CommandHandler handler = it->second;
00063         return handler(param, operationType);
00064     } else {
00065         return buildFrame(ExecutionResult::ERROR, 0, 0, "INVALID COMMAND");
00066     }
00067 } // end of CommandSystem group

```

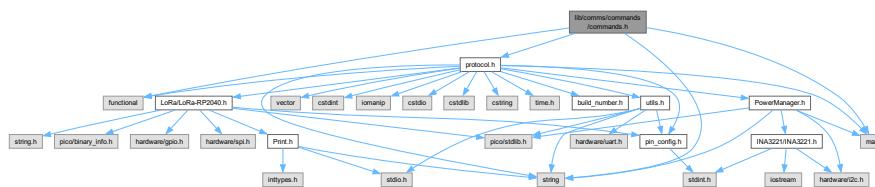
8.13 lib/comms/commands/commands.h File Reference

```

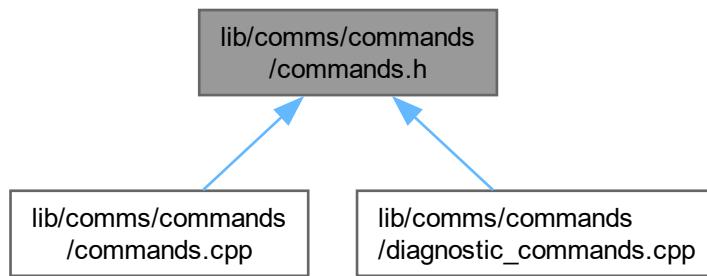
#include <string>
#include <functional>
#include <map>
#include "protocol.h"

```

Include dependency graph for commands.h:



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



Functions

- **Frame handleTime** (const std::string ¶m, OperationType operationType)
Handler for getting and setting system time.
- **Frame handleTimezoneOffset** (const std::string ¶m, OperationType operationType)
Handler for getting and setting timezone offset.
- **Frame handleClockSyncInterval** (const std::string ¶m, OperationType operationType)
Handler for getting and setting clock synchronization interval.
- **Frame handleGetLastSyncTime** (const std::string ¶m, OperationType operationType)
Handler for getting last clock sync time.
- **Frame handleListCommands** (const std::string ¶m, OperationType operationType)
Handler for listing all available commands on UART.
- **Frame handleGetBuildVersion** (const std::string ¶m, OperationType operationType)
Get firmware build version.
- **Frame handleGetCommandsTimestamp** (const std::string ¶m, OperationType operationType)
- **Frame handleEnterBootloaderMode** (const std::string ¶m, OperationType operationType)
Reboot system to USB firmware loader.
- **Frame handleGPSPowerStatus** (const std::string ¶m, OperationType operationType)
Handler for controlling GPS module power state.
- **Frame handleEnableGPSTransparentMode** (const std::string ¶m, OperationType operationType)
Handler for enabling GPS transparent mode (UART pass-through)
- **Frame handleGetRMCDData** (const std::string ¶m, OperationType operationType)
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.

- **Frame handleGetGGAData** (const std::string ¶m, OperationType operationType)
Handler for retrieving GPS GGA (Global Positioning System Fix Data) data.
- **Frame handleGetPowerManagerIDs** (const std::string ¶m, OperationType operationType)
Handler for retrieving Power Manager IDs.
- **Frame handleGetVoltageBattery** (const std::string ¶m, OperationType operationType)
Handler for getting battery voltage.
- **Frame handleGetVoltage5V** (const std::string ¶m, OperationType operationType)
Handler for getting 5V rail voltage.
- **Frame handleGetCurrentChargeUSB** (const std::string ¶m, OperationType operationType)
Handler for getting USB charge current.
- **Frame handleGetCurrentChargeSolar** (const std::string ¶m, OperationType operationType)
Handler for getting solar panel charge current.
- **Frame handleGetCurrentChargeTotal** (const std::string ¶m, OperationType operationType)
Handler for getting total charge current.
- **Frame handleGetCurrentDraw** (const std::string ¶m, OperationType operationType)
Handler for getting system current draw.
- **Frame handleGetLastEvents** (const std::string ¶m, OperationType operationType)
Handler for retrieving last N events from the event log.
- **Frame handleGetEventCount** (const std::string ¶m, OperationType operationType)
Handler for getting total number of events in the log.
- **Frame executeCommand** (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.

8.13.1 Function Documentation

8.13.1.1 handleGetCommandsTimestamp()

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

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 handleTime(const std::string& param, OperationType operationType);
00012 Frame handleTimezoneOffset(const std::string& param, OperationType operationType);
00013 Frame handleClockSyncInterval(const std::string& param, OperationType operationType);
00014 Frame handleGetLastSyncTime(const std::string& param, OperationType operationType);
```

```

00015 // DIAG
00016 Frame handleListCommands(const std::string& param, OperationType operationType);
00017 Frame handleGetBuildVersion(const std::string& param, OperationType operationType);
00018 Frame handleGetCommandsTimestamp(const std::string& param, OperationType operationType);
00019 Frame handleEnterBootloaderMode(const std::string& param, OperationType operationType);
00020 Frame handleGetGGAData(const std::string& param, OperationType operationType);
00021
00022 // GPS
00023 Frame handleGPSPowerStatus(const std::string& param, OperationType operationType);
00024 Frame handleEnableGPSTransparentMode(const std::string& param, OperationType operationType);
00025 Frame handleGetRMCDData(const std::string& param, OperationType operationType);
00026 Frame handleGetGGAData(const std::string& param, OperationType operationType);
00027
00028 // POWER
00029 Frame handleGetCurrentChargeUSB(const std::string& param, OperationType operationType);
00030 Frame handleGetVoltageBattery(const std::string& param, OperationType operationType);
00031 Frame handleGetVoltage5V(const std::string& param, OperationType operationType);
00032 Frame handleGetCurrentChargeSolar(const std::string& param, OperationType operationType);
00033 Frame handleGetCurrentChargeTotal(const std::string& param, OperationType operationType);
00034 Frame handleGetCurrentDraw(const std::string& param, OperationType operationType);
00035 Frame handleGetCurrentPower(const std::string& param, OperationType operationType);
00036
00037 // EVENT
00038 Frame handleGetLastEvents(const std::string& param, OperationType operationType);
00039 Frame handleGetEventCount(const std::string& param, OperationType operationType);
00040
00041 Frame executeCommand(uint32_t commandKey, const std::string& param, OperationType operationType);
00042 extern std::map<uint32_t, std::function<Frame(const std::string&, OperationType)>> commandHandlers;
00043
00044 #endif

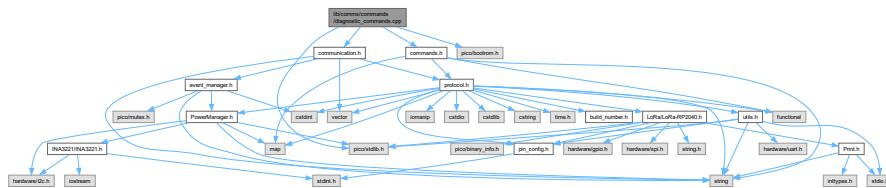
```

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 handleListCommands (const std::string ¶m, OperationType operationType)`
Handler for listing all available commands on UART.
- `Frame handleGetBuildVersion (const std::string ¶m, OperationType operationType)`
Get firmware build version.
- `Frame handleEnterBootloaderMode (const std::string ¶m, OperationType operationType)`
Reboot system to USB firmware loader.

8.16 diagnostic_commands.cpp

Go to the documentation of this file.

```

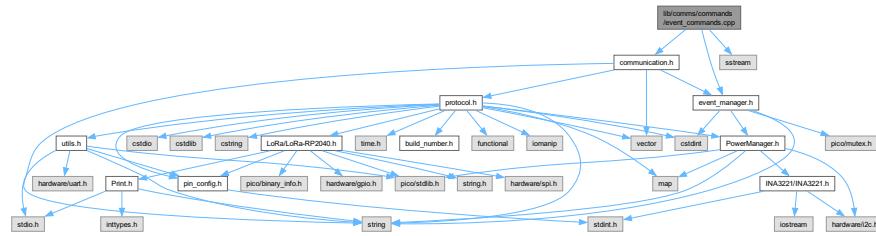
00001 #include "communication.h"
00002 #include "commands.h"
00003 #include "pico/stdlib.h"
00004 #include "pico/bootrom.h"
00005
00010
00019 Frame handleListCommands(const std::string& param, OperationType operationType) {
00020     if (!param.empty()) {
00021         return buildFrame(ExecutionResult::ERROR, 1, 0, "PARAM UNNECESSARY");
00022     }
00023
00024     if (!(operationType == OperationType::GET)) {
00025         return buildFrame(ExecutionResult::ERROR, 1, 0, "INVALID OPERATION");
00026     }
00027
00028     std::stringstream ss;
00029     for (const auto& entry : commandHandlers) {
00030         uint32_t commandKey = entry.first;
00031         uint8_t group = (commandKey >> 8) & 0xFF;
00032         uint8_t command = commandKey & 0xFF;
00033
00034         ss << "Group: " << static_cast<int>(group)
00035             << ", Command: " << static_cast<int>(command) << "\n";
00036     }
00037
00038     std::string commandList = ss.str();
00039     uartPrint(commandList, true); // Print to UART
00040
00041     return buildFrame(ExecutionResult::SUCCESS, 1, 0, "Commands listed on UART");
00042 }
00043
00052 Frame handleGetBuildVersion(const std::string& param, OperationType operationType) {
00053     if (!param.empty()) {
00054         return buildFrame(ExecutionResult::ERROR, 1, 1, "PARAM UNECESSARY");
00055     }
00056     if (operationType == OperationType::GET) {
00057         return buildFrame(ExecutionResult::SUCCESS, 1, 1, std::to_string(BUILD_NUMBER));
00058     }
00059     return buildFrame(ExecutionResult::ERROR, 1, 1, "INVALID OPERATION");
00060 }
00061
00070 Frame handleEnterBootloaderMode(const std::string& param, OperationType operationType) {
00071     if (!param.empty()) {
00072         return buildFrame(ExecutionResult::ERROR, 1, 9, "PARAM UNNECESSARY");
00073     }
00074
00075     if (operationType != OperationType::SET) {
00076         return buildFrame(ExecutionResult::ERROR, 1, 9, "INVALID OPERATION");
00077     }
00078
00079     // Build the success frame *before* resetting
00080     Frame successFrame = buildFrame(ExecutionResult::SUCCESS, 1, 9, "REBOOT BOOTSEL");
00081
00082     // Send the success frame
00083     uartPrint("Sending BOOTSEL confirmation...");
00084     sendFrame(successFrame); // Assuming you have a sendFrame function
00085
00086     // Delay to ensure the frame is sent
00087     sleep_ms(100);
00088
00089     uartPrint("Entering BOOTSEL mode...");
00090     reset_usb_boot(0, 0); // Trigger BOOTSEL mode
00091
00092     // The code will never reach here because the Pico will reset
00093     return buildFrame(ExecutionResult::SUCCESS, 1, 9, "Entering BOOTSEL mode");
00094 }
00095

```

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

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

Include dependency graph for event_commands.cpp:



Functions

- **Frame handleGetLastEvents** (const std::string ¶m, OperationType operationType)
Handler for retrieving last N events from the event log.
 - **Frame handleGetEventCount** (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 handleGetLastEvents(const std::string& param, OperationType operationType) {
00030     if (operationType != OperationType::GET) {
00031         return buildFrame(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 buildFrame(ExecutionResult::ERROR, 5, 1, "INVALID COUNT");
00040             }
00041         } catch (...) {
00042             return buildFrame(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.getEventCount();
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.getEvent(available - 1 - i);
00055         // Format: IIIITTTTTTGSEE
00056         // IIII: 16-bit ID (4 hex chars)
00057         // TTTTTTTT: 32-bit timestamp (8 hex chars)
00058         // GG: 8-bit group (2 hex chars)
00059         // EE: 8-bit event (2 hex chars)
00060         ss << std::setw(4) << event.id
00061             << std::setw(8) << event.timestamp
00062             << std::setw(2) << static_cast<int>(event.group)
00063             << std::setw(2) << static_cast<int>(event.event);
00064         if (i < toReturn - 1) ss << "-";
00065     }
00066
00067     return buildFrame(ExecutionResult::SUCCESS, 5, 1, ss.str());
00068 }
```

```

00069
00070
00082 Frame handleGetEventCount(const std::string& param, OperationType operationType) {
00083     if (operationType != OperationType::GET || !param.empty()) {
00084         return buildFrame(ExecutionResult::ERROR, 5, 2, "INVALID REQUEST");
00085     }
00086
00087     return buildFrame(ExecutionResult::SUCCESS, 5, 2,
00088         std::to_string(eventManager.getEventCount()));
00089 } // end of EventCommands group

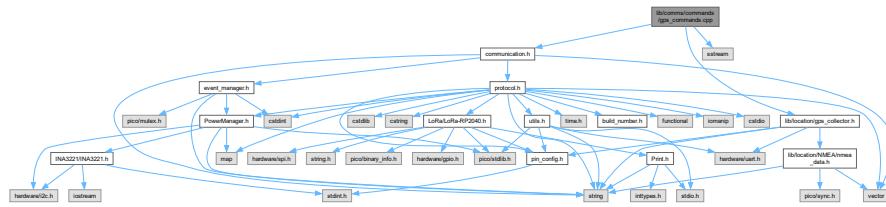
```

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

```

#include "communication.h"
#include "lib/location/gps_collector.h"
#include <iostream>
Include dependency graph for gps_commands.cpp:

```



Functions

- **Frame handleGPSPowerStatus (const std::string ¶m, OperationType operationType)**
Handler for controlling GPS module power state.
- **Frame handleEnableGPSTransparentMode (const std::string ¶m, OperationType operationType)**
Handler for enabling GPS transparent mode (UART pass-through)
- **Frame handleGetRMCDData (const std::string ¶m, OperationType operationType)**
Handler for retrieving GPS RMC (Recommended Minimum Navigation) data.
- **Frame handleGetGGAData (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 <iostream> // Include for stringstream
00004
00010
00026 Frame handleGPSPowerStatus(const std::string& param, OperationType operationType) {
00027     if (!(operationType == OperationType::GET || operationType == OperationType::SET)) {
00028         return buildFrame(ExecutionResult::ERROR, 7, 1, "INVALID OPERATION");
00029     }
00030
00031     if (operationType == OperationType::SET) {
00032         if (param.empty()) {
00033             return buildFrame(ExecutionResult::ERROR, 7, 1, "PARAM REQUIRED");
00034         }
00035
00036         try {
00037             int powerStatus = std::stoi(param);
00038             if (powerStatus != 0 && powerStatus != 1) {

```

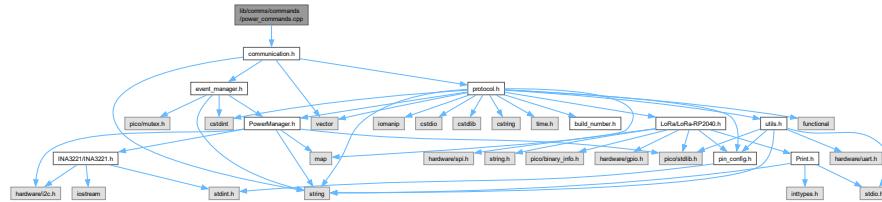
```

00039             return buildFrame(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     } catch (...) {
00045         return buildFrame(ExecutionResult::ERROR, 7, 1, "INVALID PARAMETER FORMAT");
00046     }
00047 }
00048
00049 // GET operation
00050 if (!param.empty()) {
00051     return buildFrame(ExecutionResult::ERROR, 7, 1, "PARAM UNNECESSARY");
00052 }
00053
00054 bool powerStatus = gpio_get(GPS_POWER_ENABLE_PIN);
00055 return buildFrame(ExecutionResult::SUCCESS, 7, 1, std::to_string(powerStatus));
00056 }
00057
00058
00074 Frame handleEnableGPSTransparentMode(const std::string& param, OperationType operationType) {
00075     // Validate operation type
00076     if (!(operationType == OperationType::SET)) {
00077         uartPrint("GET operation not allowed for EnableGPSTransparentMode");
00078         return buildFrame(ExecutionResult::ERROR, 7, 2, "NOT ALLOWED");
00079     }
00080
00081     // Parse and validate timeout parameter
00082     uint32_t timeoutMs;
00083     try {
00084         timeoutMs = param.empty() ? 60000u : std::stoul(param) * 1000;
00085     } catch (...) {
00086         return buildFrame(ExecutionResult::ERROR, 7, 2, "INVALID TIMEOUT FORMAT");
00087     }
00088
00089     // Setup UART parameters and exit sequence
00090     const std::string EXIT_SEQUENCE = "##EXIT##";
00091     std::string inputBuffer;
00092     bool exitRequested = false;
00093     uint32_t originalBaudRate = DEBUG_UART_BAUD_RATE;
00094     uint32_t gpsBaudRate = GPS_UART_BAUD_RATE;
00095     uint32_t startTime = to_ms_since_boot(get_absolute_time());
00096
00097     // Log start of transparent mode
00098     EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_START);
00099
00100     // Print startup message
00101     std::string message = "Entering GPS Serial Pass-Through Mode @" +
00102         std::to_string(gpsBaudRate) + " for " +
00103         std::to_string(timeoutMs/1000) + "s\r\n" +
00104         "Send " + EXIT_SEQUENCE + " to exit";
00105     uartPrint(message);
00106
00107     // Allow time for message to be sent before baudrate change
00108     sleep_ms(10);
00109
00110     // Switch to GPS baudrate
00111     uart_set_baudrate(DEBUG_UART_PORT, gpsBaudRate);
00112
00113     // Main transparent mode loop
00114     while (!exitRequested) {
00115         while (uart_is_readable(DEBUG_UART_PORT)) {
00116             char ch = uart_getc(DEBUG_UART_PORT);
00117
00118             inputBuffer += ch;
00119             if (inputBuffer.length() > EXIT_SEQUENCE.length()) {
00120                 inputBuffer = inputBuffer.substr(1);
00121             }
00122
00123             if (inputBuffer == EXIT_SEQUENCE) {
00124                 exitRequested = true;
00125                 break;
00126             }
00127
00128             if (inputBuffer != EXIT_SEQUENCE.substr(0, inputBuffer.length())) {
00129                 uart_write_blocking(GPS_UART_PORT,
00130                     reinterpret_cast<const uint8_t*>(&ch), 1);
00131             }
00132         }
00133
00134         while (uart_is_readable(GPS_UART_PORT)) {
00135             char gpsByte = uart_getc(GPS_UART_PORT);
00136             uart_write_blocking(DEBUG_UART_PORT,
00137                 reinterpret_cast<const uint8_t*>(&gpsByte), 1);
00138     }
00139 }
```

```
00140     if (to_ms_since_boot(get_absolute_time()) - startTime >= timeoutMs) {
00141         break;
00142     }
00143 }
00144
00145     uart_set_baudrate(DEBUG_UART_PORT, originalBaudRate);
00146
00147     sleep_ms(10);
00148
00149     EventEmitter::emit(EventGroup::GPS, GPSEvent::PASS_THROUGH_END);
00150
00151     std::string exitReason = exitRequested ? "USER_EXIT" : "TIMEOUT";
00152     std::string response = "GPS UART BRIDGE EXIT: " + exitReason;
00153     uartPrint(response);
00154
00155     return buildFrame(ExecutionResult::SUCCESS, 7, 2, response);
00156 }
00157
00158
00171 Frame handleGetRMCData(const std::string& param, OperationType operationType) {
00172     if (operationType != OperationType::GET) {
00173         return buildFrame(ExecutionResult::ERROR, 7, 3, "INVALID OPERATION");
00174     }
00175
00176     if (!param.empty()) {
00177         return buildFrame(ExecutionResult::ERROR, 7, 3, "PARAM UNNECESSARY");
00178     }
00179
00180     std::vector<std::string> tokens = nmea_data.getRmcTokens();
00181     if (tokens.empty()) {
00182         return buildFrame(ExecutionResult::ERROR, 7, 3, "NO RMC DATA");
00183     }
00184
00185     // Join tokens with commas to create the response
00186     std::stringstream ss;
00187     for (size_t i = 0; i < tokens.size(); ++i) {
00188         ss << tokens[i];
00189         if (i < tokens.size() - 1) {
00190             ss << ",";
00191         }
00192     }
00193
00194     return buildFrame(ExecutionResult::SUCCESS, 7, 3, ss.str());
00195 }
00196
00197
00210 Frame handleGetGGAData(const std::string& param, OperationType operationType) {
00211     if (operationType != OperationType::GET) {
00212         return buildFrame(ExecutionResult::ERROR, 7, 4, "INVALID OPERATION");
00213     }
00214
00215     if (!param.empty()) {
00216         return buildFrame(ExecutionResult::ERROR, 7, 4, "PARAM UNNECESSARY");
00217     }
00218
00219     std::vector<std::string> tokens = nmea_data.getGgaTokens();
00220     if (tokens.empty()) {
00221         return buildFrame(ExecutionResult::ERROR, 7, 4, "NO GGA DATA");
00222     }
00223
00224     // Join tokens with commas to create the response
00225     std::stringstream ss;
00226     for (size_t i = 0; i < tokens.size(); ++i) {
00227         ss << tokens[i];
00228         if (i < tokens.size() - 1) {
00229             ss << ",";
00230         }
00231     }
00232
00233     return buildFrame(ExecutionResult::SUCCESS, 7, 4, ss.str());
00234 } // 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 handleGetPowerManagerIDs (const std::string ¶m, OperationType operationType)**
Handler for retrieving Power Manager IDs.
- **Frame handleGetVoltageBattery (const std::string ¶m, OperationType operationType)**
Handler for getting battery voltage.
- **Frame handleGetVoltage5V (const std::string ¶m, OperationType operationType)**
Handler for getting 5V rail voltage.
- **Frame handleGetCurrentChargeUSB (const std::string ¶m, OperationType operationType)**
Handler for getting USB charge current.
- **Frame handleGetCurrentChargeSolar (const std::string ¶m, OperationType operationType)**
Handler for getting solar panel charge current.
- **Frame handleGetCurrentChargeTotal (const std::string ¶m, OperationType operationType)**
Handler for getting total charge current.
- **Frame handleGetCurrentDraw (const std::string ¶m, OperationType operationType)**
Handler for getting system current draw.

8.22 power_commands.cpp

Go to the documentation of this file.

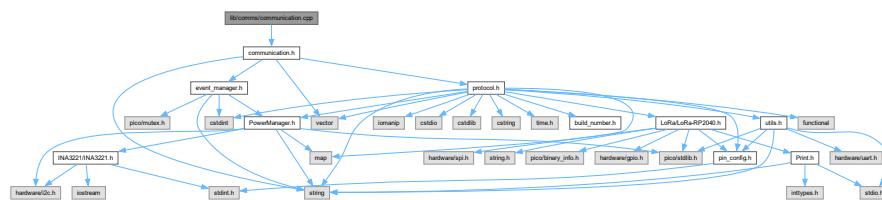
```
00001 #include "communication.h"
00002
00003
00009
00021 Frame handleGetPowerManagerIDs(const std::string& param, OperationType operationType) {
00022     if (!param.empty()) {
00023         return buildFrame(ExecutionResult::ERROR, 2, 0, "PARAM UNNECESSARY");
00024     }
00025
00026     if (!(operationType == OperationType::GET)) {
00027         return buildFrame(ExecutionResult::ERROR, 2, 0, "INVALID OPERATION");
00028     }
00029
00030     extern PowerManager powerManager;
00031     std::string powerManagerIDS = powerManager.readIDs();
00032     return buildFrame(ExecutionResult::SUCCESS, 2, 0, powerManagerIDS);
00033 }
00034
00035
00047 Frame handleGetVoltageBattery(const std::string& param, OperationType operationType) {
00048     if (!param.empty()) {
00049         return buildFrame(ExecutionResult::ERROR, 2, 2, "PARAM UNNECESSARY");
00050     }
00051     if (!(operationType == OperationType::GET)) {
```

```
00053     uartPrint("SET operation not allowed for GetVoltageBattery");
00054     return buildFrame(ExecutionResult::ERROR, 2, 2, "NOT ALLOWED");
00055 }
00056
00057     uartPrint("Getting battery voltage");
00058     extern PowerManager powerManager;
00059     float voltage = powerManager.getVoltageBattery();
00060     return buildFrame(ExecutionResult::SUCCESS, 2, 2, std::to_string(voltage), ValueUnit::VOLT);
00061 }
00062
00063
00075 Frame handleGetVoltage5V(const std::string& param, OperationType operationType) {
00076     if (!param.empty()) {
00077         return buildFrame(ExecutionResult::ERROR, 2, 3, "PARAM UNNECESSARY");
00078     }
00079
00080     if (!(operationType == OperationType::GET)) {
00081         uartPrint("SET operation not allowed for GetVoltage5V");
00082         return buildFrame(ExecutionResult::ERROR, 2, 3, "NOT ALLOWED");
00083     }
00084
00085     uartPrint("Getting 5V voltage");
00086     extern PowerManager powerManager;
00087     float voltage = powerManager.getVoltage5V();
00088     return buildFrame(ExecutionResult::SUCCESS, 2, 3, std::to_string(voltage), ValueUnit::VOLT);
00089 }
00090
00091
00103 Frame handleGetCurrentChargeUSB(const std::string& param, OperationType operationType) {
00104     if (!param.empty()) {
00105         return buildFrame(ExecutionResult::ERROR, 2, 4, "PARAM UNNECESSARY");
00106     }
00107
00108     if (!(operationType == OperationType::GET)) {
00109         uartPrint("SET operation not allowed for GetCurrentChargeUSB");
00110         return buildFrame(ExecutionResult::ERROR, 2, 4, "NOT ALLOWED");
00111     }
00112
00113     uartPrint("Getting USB charge current");
00114     extern PowerManager powerManager;
00115     float chargeCurrent = powerManager.getCurrentChargeUSB();
00116     return buildFrame(ExecutionResult::SUCCESS, 2, 4, std::to_string(chargeCurrent),
00117     ValueUnit::MILIAMP);
00118 }
00119
00120
00131 Frame handleGetCurrentChargeSolar(const std::string& param, OperationType operationType) {
00132     if (!param.empty()) {
00133         return buildFrame(ExecutionResult::ERROR, 2, 5, "PARAM UNNECESSARY");
00134     }
00135
00136     if (!(operationType == OperationType::GET)) {
00137         uartPrint("SET operation not allowed for GetCurrentChargeSolar");
00138         return buildFrame(ExecutionResult::ERROR, 2, 5, "NOT ALLOWED");
00139     }
00140
00141     uartPrint("Getting solar charge current");
00142     extern PowerManager powerManager;
00143     float chargeCurrent = powerManager.getCurrentChargeSolar();
00144     return buildFrame(ExecutionResult::SUCCESS, 2, 5, std::to_string(chargeCurrent),
00145     ValueUnit::MILIAMP);
00146 }
00147
00148
00159 Frame handleGetCurrentChargeTotal(const std::string& param, OperationType operationType) {
00160     if (!param.empty()) {
00161         return buildFrame(ExecutionResult::ERROR, 2, 6, "PARAM UNNECESSARY");
00162     }
00163
00164     if (!(operationType == OperationType::GET)) {
00165         uartPrint("SET operation not allowed for GetCurrentChargeTotal");
00166         return buildFrame(ExecutionResult::ERROR, 2, 6, "NOT ALLOWED");
00167     }
00168
00169     uartPrint("Getting total charge current");
00170     extern PowerManager powerManager;
00171     float chargeCurrent = powerManager.getCurrentChargeTotal();
00172     return buildFrame(ExecutionResult::SUCCESS, 2, 6, std::to_string(chargeCurrent),
00173     ValueUnit::MILIAMP);
00174 }
00175
00176
00187 Frame handleGetCurrentDraw(const std::string& param, OperationType operationType) {
00188     if (!param.empty()) {
00189         return buildFrame(ExecutionResult::ERROR, 2, 7, "PARAM UNNECESSARY");
00190     }
00191 }
```

```
00192     if (!operationType == OperationType::GET)) {
00193         uartPrint("SET operation not allowed for GetCurrentDraw");
00194         return buildFrame(ExecutionResult::ERROR, 2, 7, "NOT ALLOWED");
00195     }
00196
00197     uartPrint("Getting current draw");
00198     extern PowerManager powerManager;
00199     float currentDraw = powerManager.getCurrentDraw();
00200     return buildFrame(ExecutionResult::SUCCESS, 2, 7, std::to_string(currentDraw),
ValueUnit::MILLIAMP);
00201 } // end of PowerCommands group
```

8.23 lib/comms/communication.cpp File Reference

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



Functions

- **bool initializeRadio ()**
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

8.23.1 Function Documentation

8.23.1.1 initializeRadio()

```
bool initializeRadio ()
```

Initializes the LoRa radio module.

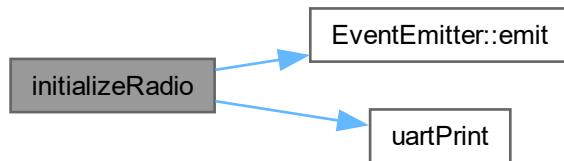
Returns

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:



8.23.2 Variable Documentation

8.23.2.1 outgoing

```
string outgoing
```

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

8.23.2.2 msgCount

```
uint8_t msgCount = 0
```

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

8.23.2.3 lastSendTime

```
long lastSendTime = 0
```

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

8.23.2.4 lastReceiveTime

```
long lastReceiveTime = 0
```

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

8.23.2.5 lastPrintTime

```
long lastPrintTime = 0
```

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

8.23.2.6 interval

```
unsigned long interval = 0
```

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

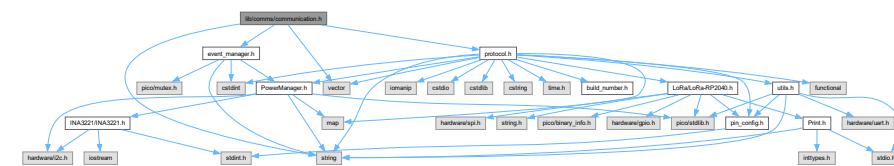
8.24 communication.cpp

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

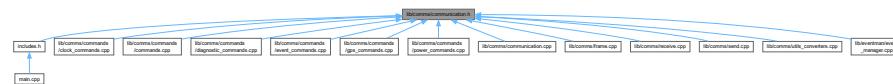
```
00001 #include "communication.h"
00002
00003 string outgoing;
00004 uint8_t msgCount = 0;
00005 long lastSendTime = 0;
00006 long lastReceiveTime = 0;
00007 long lastPrintTime = 0;
00008 unsigned long interval = 0;
00009
00010
00011 bool initializeRadio() {
00012     LoRa.setPins(csPin, resetPin,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

8.25 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 `initializeRadio ()`
Initializes the LoRa radio module.
 - void `sendMessage (std::string outgoing)`
 - void `sendLargePacket (const uint8_t *data, size_t length)`
Sends a large packet using LoRa.
 - void `onReceive (int packetSize)`
Callback function for handling received LoRa packets.
 - void `handleUartInput ()`
Handles UART input.
 - void `processFrameData (const std::string &data)`
Executes a command based on the command key and the parameter.
 - void `handleCommandFrame (const Frame &frame)`
 - Frame `executeCommand (uint32_t commandKey, const std::string ¶m, OperationType operationType)`
Executes a command based on its key.
 - void `sendEventRegister ()`
Sends the event register value to the ground station.
 - std::string `encodeFrame (const Frame &frame)`
Encodes a `Frame` instance into a string.
 - Frame `decodeFrame (const std::string &data)`
Converts a string into a `Frame` instance.
 - Frame `buildFrame (ExecutionResult result, uint8_t group, uint8_t command, const std::string &value, const ValueUnit unitType=ValueUnit::UNDEFINED)`
Builds a `Frame` instance.
 - std::string `determineUnit (uint8_t group, uint8_t command)`
 - void `sendFrame (const Frame &frame)`
Sends a `Frame` using LoRa.

8.25.1 Function Documentation

8.25.1.1 initializeRadio()

```
bool initializeRadio ()
```

Initializes the LoRa radio module.

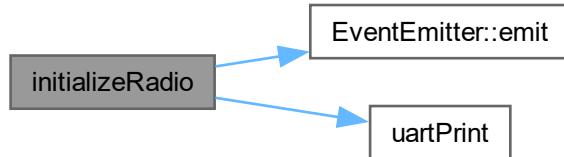
Returns

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:



8.25.1.2 sendMessage()

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

Here is the caller graph for this function:



8.25.1.3 sendLargePacket()

```
void sendLargePacket (
    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 55 of file [send.cpp](#).

8.25.1.4 onReceive()

```
void onReceive (
    int packetSize)
```

Callback function for handling received LoRa packets.

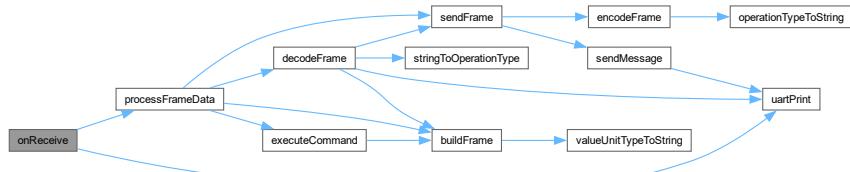
Parameters

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

Reads the received LoRa packet, extracts metadata, validates the destination 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:



8.25.1.5 handleUartInput()

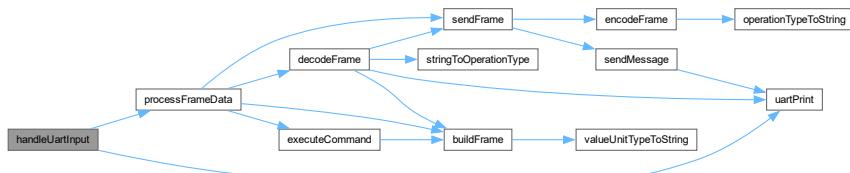
```
void handleUartInput ()
```

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 77 of file [receive.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.25.1.6 processFrameData()

```
void processFrameData (
    const std::string & data)
```

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

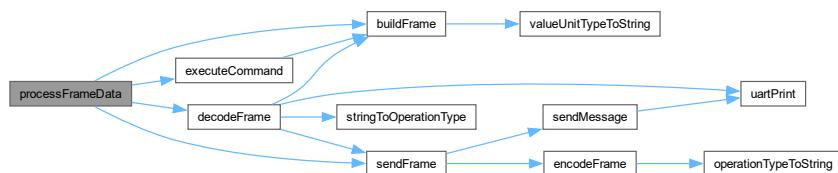
Parameters

| | |
|-------------|----------------------------------|
| <i>data</i> | The Frame data in string format. |
|-------------|----------------------------------|

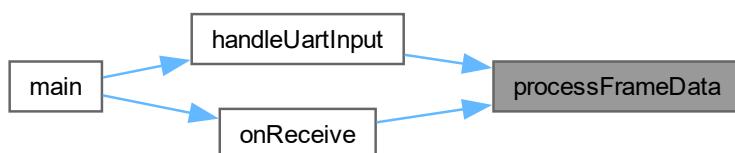
Decodes the frame data, extracts the command key, and executes the corresponding command. Sends the response frame. If an error occurs, an error frame is built and sent.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.25.1.7 handleCommandFrame()

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

8.25.1.8 sendEventRegister()

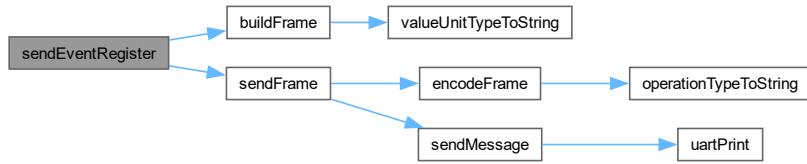
```
void sendEventRegister ()
```

Sends the event register value to the ground station.

This function is called in the main loop to send the current value of the event register.

Definition at line 119 of file [frame.cpp](#).

Here is the call graph for this function:



8.25.1.9 encodeFrame()

```
std::string encodeFrame (
    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.

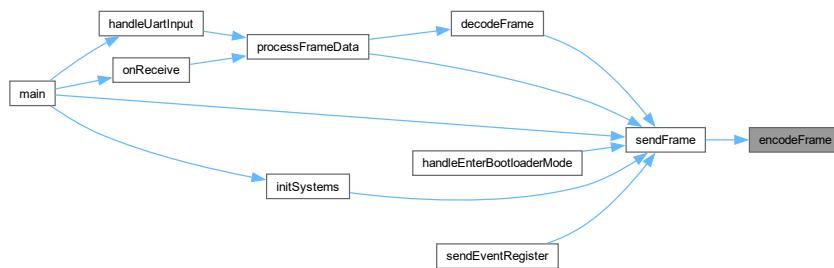
The encoded string includes the frame direction, operation type, group, command, value, and unit, all delimited by the DELIMITER character. The string is encapsulated by FRAME_BEGIN and FRAME_END.

Definition at line 19 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.25.1.10 decodeFrame()

```
Frame decodeFrame (
    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.

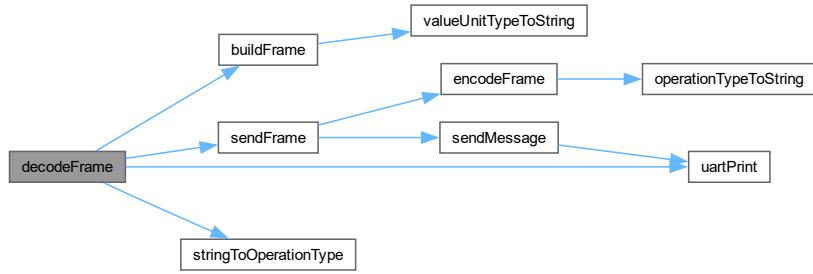
Exceptions

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

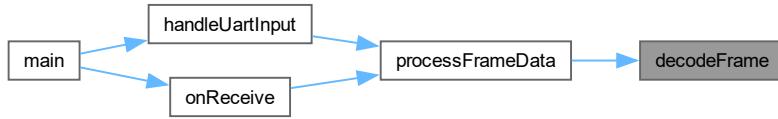
Parses the input string, extracting the frame direction, operation type, group, command, value, and unit. If an error occurs during parsing, an error message is printed to the UART, an error frame is built and sent, and the exception is re-thrown.

Definition at line 44 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.25.1.11 `buildFrame()`

```

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

Builds a `Frame` instance.

Parameters

| | |
|-----------------------|-----------------------|
| <code>result</code> | The execution result. |
| <code>group</code> | The group ID. |
| <code>command</code> | The command ID. |
| <code>value</code> | The value. |
| <code>unitType</code> | The value unit type. |

Returns

The [Frame](#) instance.

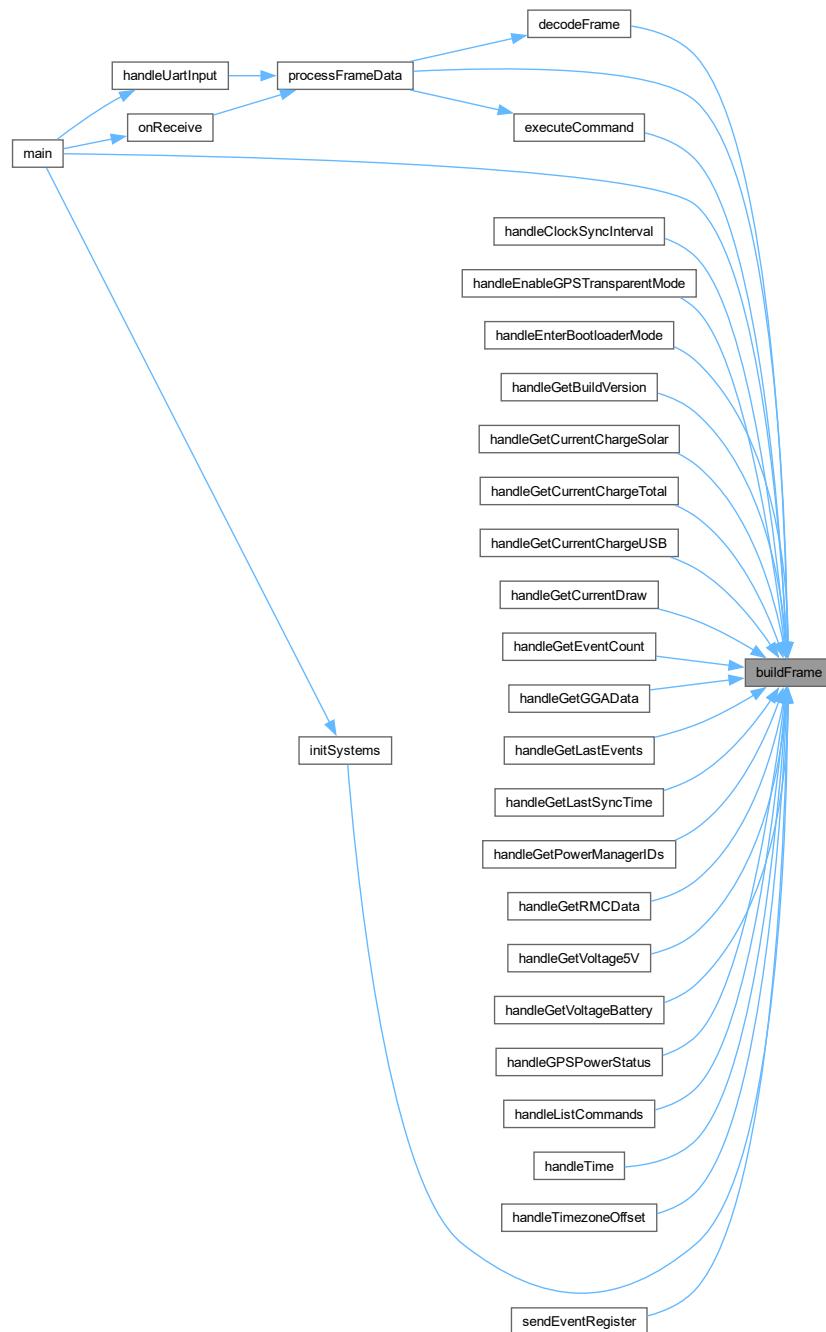
Constructs a [Frame](#) instance based on the provided parameters. The frame direction and operation type are set based on the execution result.

Definition at line 146 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.25.1.12 `determineUnit()`

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

8.25.1.13 sendFrame()

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

Sends a [Frame](#) using LoRa.

Parameters

| | |
|--------------|------------------------------------|
| <i>frame</i> | The Frame to send. |
|--------------|------------------------------------|

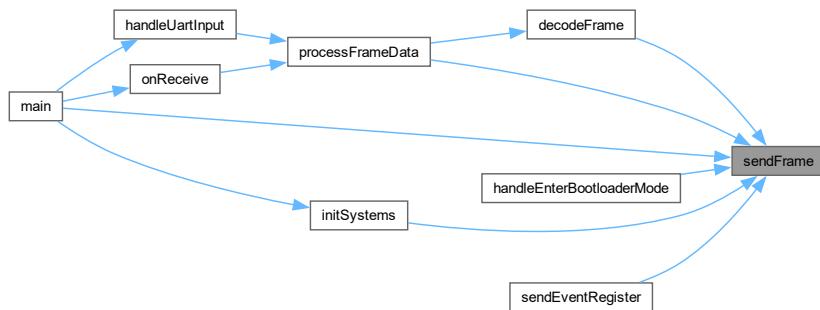
Encodes the [Frame](#) into a string and sends it using the [sendMessage](#) function.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.26 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 initializeRadio();
00010 void sendMessage(std::string outgoing);
00011 void sendLargePacket(const uint8_t* data, size_t length);
00012 void onReceive(int packetSize);
00013 void handleUartInput();
00014 void processFrameData(const std::string& data);
00015
00016
00017 void handleCommandFrame(const Frame& frame);
00018 Frame executeCommand(uint32_t commandKey, const std::string& param, OperationType operationType);
00019 void sendEventRegister();
00020
00021
00022 std::string encodeFrame(const Frame& frame);
00023 Frame decodeFrame(const std::string& data);
00024
00025 Frame buildFrame(ExecutionResult result, uint8_t group, uint8_t command, const std::string& value,
    const ValueUnit unitType = ValueUnit::UNDEFINED);
00026 std::string determineUnit(uint8_t group, uint8_t command);
00027
00028 void sendMessage(std::string outgoing);
00029 void sendFrame(const Frame& frame);
00030 void sendLargePacket(const uint8_t* data, size_t length);
00031
00032
00033 void onReceive(int packetSize);
00034 void handleUartInput();
00035 void processFrameData(const std::string& data);
00036
00037
00038 #endif

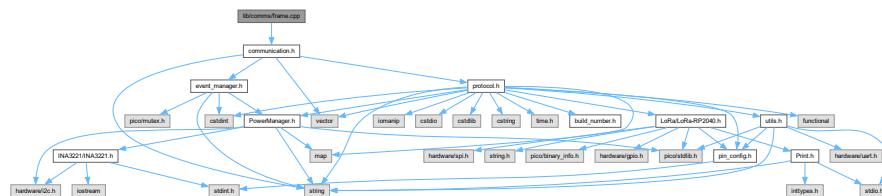
```

8.27 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 **encodeFrame** (const Frame &frame)

Encodes a Frame instance into a string.
- Frame **decodeFrame** (const std::string &data)

Converts a string into a Frame instance.
- void **processFrameData** (const std::string &data)

Executes a command based on the command key and the parameter.
- void **sendEventRegister** ()

Sends the event register value to the ground station.
- Frame **buildFrame** (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

8.27.1 Detailed Description

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

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

8.27.2 Typedef Documentation

8.27.2.1 CommandHandler

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

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

8.27.3 Function Documentation

8.27.3.1 encodeFrame()

```
std::string encodeFrame (
    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.

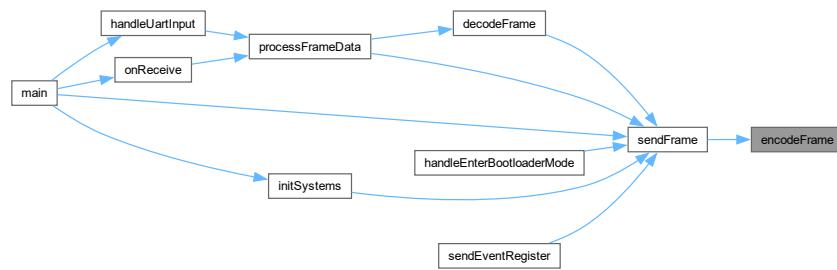
The encoded string includes the frame direction, operation type, group, command, value, and unit, all delimited by the DELIMITER character. The string is encapsulated by FRAME_BEGIN and FRAME_END.

Definition at line 19 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.27.3.2 decodeFrame()

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

Converts a string into a `Frame` instance.

Parameters

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

Returns

The `Frame` instance.

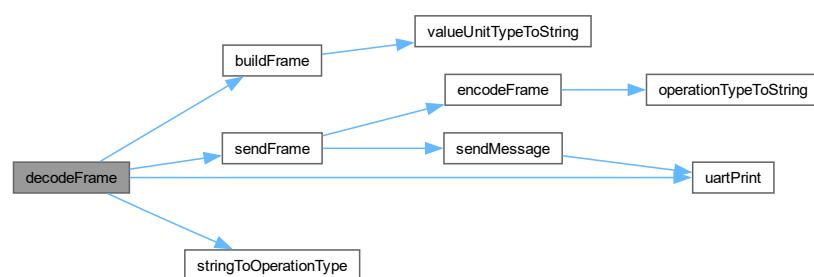
Exceptions

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

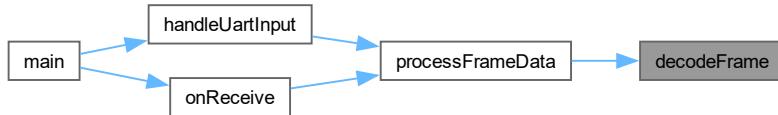
Parses the input string, extracting the frame direction, operation type, group, command, value, and unit. If an error occurs during parsing, an error message is printed to the UART, an error frame is built and sent, and the exception is re-thrown.

Definition at line 44 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.27.3.3 processFrameData()

```
void processFrameData (
    const std::string & data)
```

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

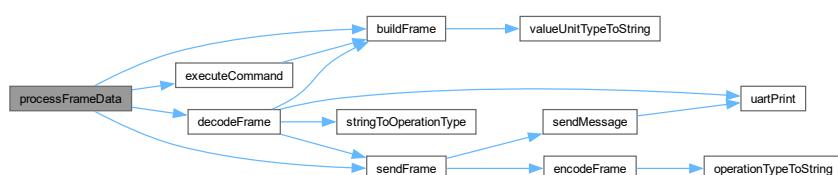
Parameters

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

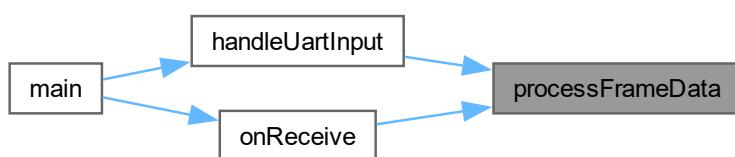
Decodes the frame data, extracts the command key, and executes the corresponding command. Sends the response frame. If an error occurs, an error frame is built and sent.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.27.3.4 sendEventRegister()

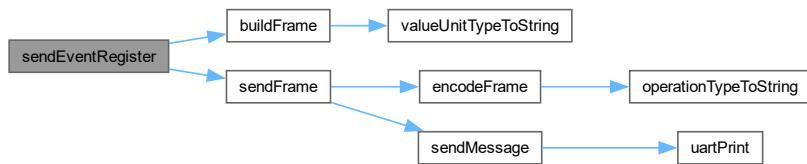
```
void sendEventRegister ()
```

Sends the event register value to the ground station.

This function is called in the main loop to send the current value of the event register.

Definition at line 119 of file [frame.cpp](#).

Here is the call graph for this function:



8.27.3.5 buildFrame()

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

Builds a [Frame](#) instance.

Parameters

| | |
|-----------------------|-----------------------|
| <code>result</code> | The execution result. |
| <code>group</code> | The group ID. |
| <code>command</code> | The command ID. |
| <code>value</code> | The value. |
| <code>unitType</code> | The value unit type. |

Returns

The [Frame](#) instance.

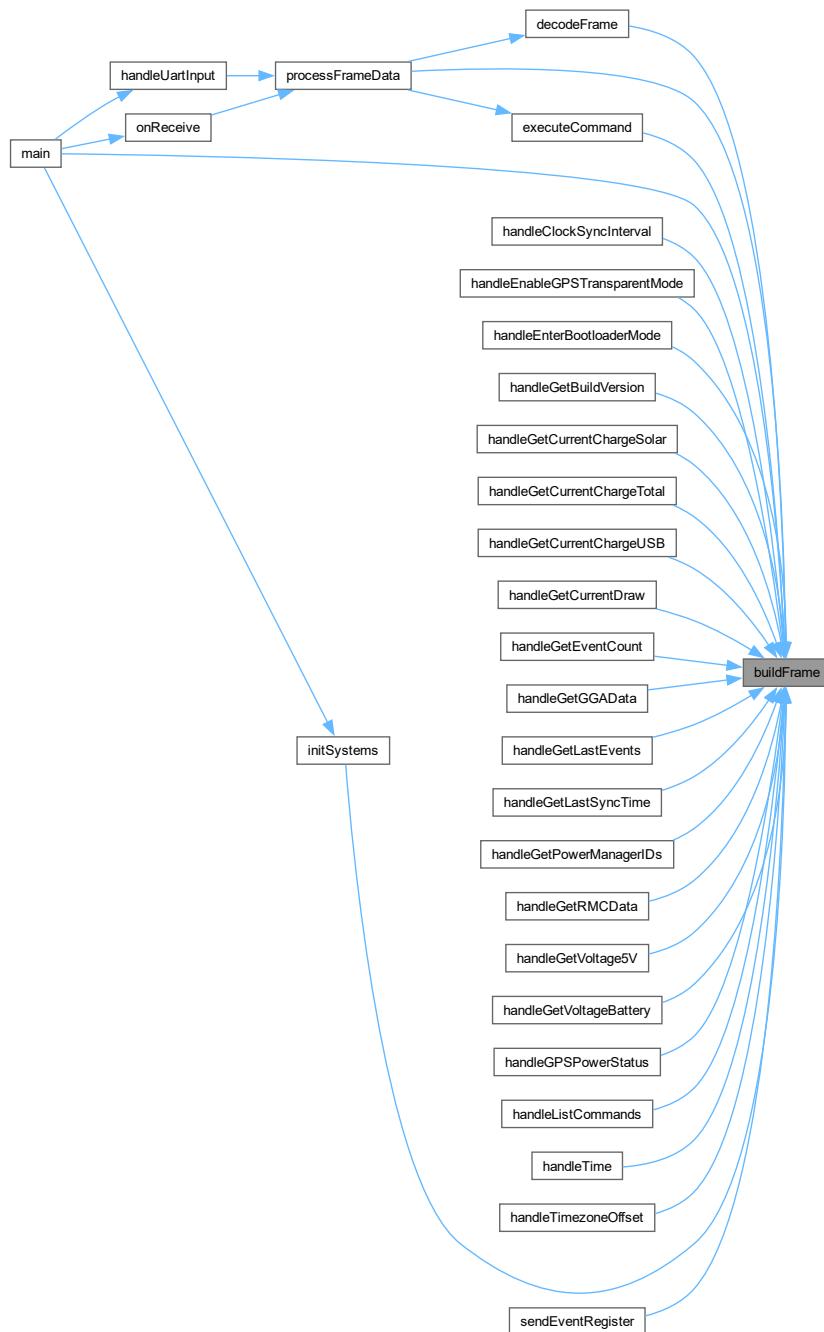
Constructs a [Frame](#) instance based on the provided parameters. The frame direction and operation type are set based on the execution result.

Definition at line 146 of file [frame.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.27.4 Variable Documentation

8.27.4.1 eventRegister

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

8.28 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 encodeFrame(const Frame& frame) {
00010     std::stringstream ss;
00011     ss << static_cast<int>(frame.direction) << DELIMITER
00012         << operationTypeToString(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 decodeFrame(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 = stringToOperationType(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         uartPrint("Frame error: " + std::string(e.what()));
00087         Frame errorFrame = buildFrame(ExecutionResult::ERROR, 0, 0, e.what());
00088         sendFrame(errorFrame);
00089         throw;
00090     }
00091 }
00092
00093
00094
00095
00096
00097
00098
00099
00100 void processFrameData(const std::string& data) {
00101     try {
00102         Frame frame = decodeFrame(data);
00103         uint32_t commandKey = (static_cast<uint32_t>(frame.group) << 8) |
00104             static_cast<uint32_t>(frame.command);
00105         Frame responseFrame = executeCommand(commandKey, frame.value, frame.operationType);
00106         sendFrame(responseFrame);
00107 }
```

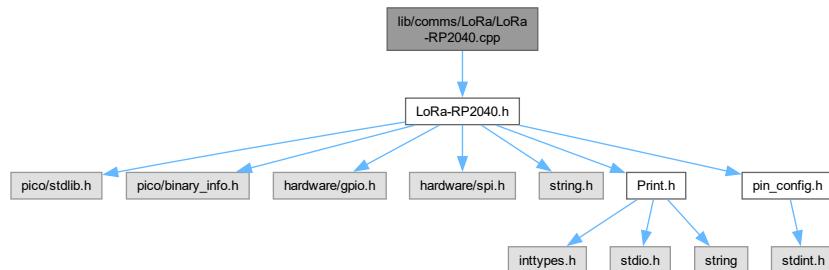
```

00108     } catch (const std::exception& e) {
00109         Frame errorFrame = buildFrame(ExecutionResult::ERROR, 0, 0, e.what());
00110         sendFrame(errorFrame);
00111     }
00112 }
00113
00114
00115 void sendEventRegister() {
00116     std::stringstream ss;
00117     ss << std::hex << std::setw(4) << std::setfill('0') << static_cast<int>(eventRegister);
00118     std::string eventValue = ss.str();
00119
00120     Frame eventFrame = buildFrame(
00121         ExecutionResult::SUCCESS, // Result: success as this is a normal status update
00122         8,                      // Group ID: 8 (EVENTS)
00123         0,                      // Command ID: 0 (EVENT_REGISTER)
00124         eventValue              // Value: event register value
00125     );
00126
00127     sendFrame(eventFrame);
00128 }
00129
00130
00131
00132 }
00133
00134
00135
00136 Frame buildFrame(ExecutionResult result, uint8_t group, uint8_t command,
00137                     const std::string& value, const ValueUnit unitType) {
00138     Frame frame;
00139     frame.header = FRAME_BEGIN;
00140     frame.footer = FRAME_END;
00141
00142     switch (result) {
00143         case ExecutionResult::SUCCESS:
00144             frame.direction = 1;
00145             frame.operationType = OperationType::ANS;
00146             frame.value = value;
00147             frame.unit = valueUnitTypeToString(unitType);
00148             break;
00149
00150         case ExecutionResult::ERROR:
00151             frame.direction = 1;
00152             frame.operationType = OperationType::ERR;
00153             frame.value = value;
00154             frame.unit = valueUnitTypeToString(ValueUnit::UNDEFINED);
00155             break;
00156
00157         case ExecutionResult::INFO:
00158             frame.direction = 1;
00159             frame.operationType = OperationType::INF;
00160             frame.value = value;
00161             frame.unit = valueUnitTypeToString(ValueUnit::UNDEFINED);
00162             break;
00163
00164     }
00165
00166     frame.group = group;
00167     frame.command = command;
00168
00169     return frame;
00170 }
00171
00172
00173
00174
00175
00176
00177
00178
00179 }

```

8.29 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

8.29.1 Macro Definition Documentation

8.29.1.1 REG_FIFO

```
#define REG_FIFO 0x00
```

Definition at line 3 of file [LoRa-RP2040.cpp](#).

8.29.1.2 REG_OP_MODE

```
#define REG_OP_MODE 0x01
```

Definition at line 4 of file [LoRa-RP2040.cpp](#).

8.29.1.3 REG_FRF_MSB

```
#define REG_FRF_MSB 0x06
```

Definition at line 5 of file [LoRa-RP2040.cpp](#).

8.29.1.4 REG_FRF_MID

```
#define REG_FRF_MID 0x07
```

Definition at line 6 of file [LoRa-RP2040.cpp](#).

8.29.1.5 REG_FRF_LSB

```
#define REG_FRF_LSB 0x08
```

Definition at line 7 of file [LoRa-RP2040.cpp](#).

8.29.1.6 REG_PA_CONFIG

```
#define REG_PA_CONFIG 0x09
```

Definition at line 8 of file [LoRa-RP2040.cpp](#).

8.29.1.7 IRQ_CAD_DETECTED_MASK

```
#define IRQ_CAD_DETECTED_MASK 0x01
```

Definition at line 9 of file [LoRa-RP2040.cpp](#).

8.29.1.8 REG_OCP

```
#define REG_OCP 0x0b
```

Definition at line 10 of file [LoRa-RP2040.cpp](#).

8.29.1.9 REG_LNA

```
#define REG_LNA 0x0c
```

Definition at line 11 of file [LoRa-RP2040.cpp](#).

8.29.1.10 IRQ_CAD_DONE_MASK

```
#define IRQ_CAD_DONE_MASK 0x04
```

Definition at line 12 of file [LoRa-RP2040.cpp](#).

8.29.1.11 REG_FIFO_ADDR_PTR

```
#define REG_FIFO_ADDR_PTR 0x0d
```

Definition at line 13 of file [LoRa-RP2040.cpp](#).

8.29.1.12 REG_FIFO_TX_BASE_ADDR

```
#define REG_FIFO_TX_BASE_ADDR 0x0e
```

Definition at line 14 of file [LoRa-RP2040.cpp](#).

8.29.1.13 REG_FIFO_RX_BASE_ADDR

```
#define REG_FIFO_RX_BASE_ADDR 0x0f
```

Definition at line 15 of file [LoRa-RP2040.cpp](#).

8.29.1.14 REG_FIFO_RX_CURRENT_ADDR

```
#define REG_FIFO_RX_CURRENT_ADDR 0x10
```

Definition at line 16 of file [LoRa-RP2040.cpp](#).

8.29.1.15 REG_IRQ_FLAGS

```
#define REG_IRQ_FLAGS 0x12
```

Definition at line 17 of file [LoRa-RP2040.cpp](#).

8.29.1.16 REG_RX_NB_BYTES

```
#define REG_RX_NB_BYTES 0x13
```

Definition at line 18 of file [LoRa-RP2040.cpp](#).

8.29.1.17 REG_PKT_SNR_VALUE

```
#define REG_PKT_SNR_VALUE 0x19
```

Definition at line 19 of file [LoRa-RP2040.cpp](#).

8.29.1.18 REG_PKT_RSSI_VALUE

```
#define REG_PKT_RSSI_VALUE 0x1a
```

Definition at line 20 of file [LoRa-RP2040.cpp](#).

8.29.1.19 REG_RSSI_VALUE

```
#define REG_RSSI_VALUE 0x1b
```

Definition at line 21 of file [LoRa-RP2040.cpp](#).

8.29.1.20 REG_MODEM_CONFIG_1

```
#define REG_MODEM_CONFIG_1 0x1d
```

Definition at line 22 of file [LoRa-RP2040.cpp](#).

8.29.1.21 REG_MODEM_CONFIG_2

```
#define REG_MODEM_CONFIG_2 0x1e
```

Definition at line 23 of file [LoRa-RP2040.cpp](#).

8.29.1.22 REG_PREAMBLE_MSB

```
#define REG_PREAMBLE_MSB 0x20
```

Definition at line 24 of file [LoRa-RP2040.cpp](#).

8.29.1.23 REG_PREAMBLE_LSB

```
#define REG_PREAMBLE_LSB 0x21
```

Definition at line 25 of file [LoRa-RP2040.cpp](#).

8.29.1.24 REG_PAYLOAD_LENGTH

```
#define REG_PAYLOAD_LENGTH 0x22
```

Definition at line [26](#) of file [LoRa-RP2040.cpp](#).

8.29.1.25 REG_MODEM_CONFIG_3

```
#define REG_MODEM_CONFIG_3 0x26
```

Definition at line [27](#) of file [LoRa-RP2040.cpp](#).

8.29.1.26 REG_FREQ_ERROR_MSB

```
#define REG_FREQ_ERROR_MSB 0x28
```

Definition at line [28](#) of file [LoRa-RP2040.cpp](#).

8.29.1.27 REG_FREQ_ERROR_MID

```
#define REG_FREQ_ERROR_MID 0x29
```

Definition at line [29](#) of file [LoRa-RP2040.cpp](#).

8.29.1.28 REG_FREQ_ERROR_LSB

```
#define REG_FREQ_ERROR_LSB 0x2a
```

Definition at line [30](#) of file [LoRa-RP2040.cpp](#).

8.29.1.29 REG_RSSI_WIDEBAND

```
#define REG_RSSI_WIDEBAND 0x2c
```

Definition at line [31](#) of file [LoRa-RP2040.cpp](#).

8.29.1.30 REG_DETECTION_OPTIMIZE

```
#define REG_DETECTION_OPTIMIZE 0x31
```

Definition at line [32](#) of file [LoRa-RP2040.cpp](#).

8.29.1.31 REG_INVERTIQ

```
#define REG_INVERTIQ 0x33
```

Definition at line [33](#) of file [LoRa-RP2040.cpp](#).

8.29.1.32 REG_DETECTION_THRESHOLD

```
#define REG_DETECTION_THRESHOLD 0x37
```

Definition at line 34 of file [LoRa-RP2040.cpp](#).

8.29.1.33 REG_SYNC_WORD

```
#define REG_SYNC_WORD 0x39
```

Definition at line 35 of file [LoRa-RP2040.cpp](#).

8.29.1.34 REG_INVERTIQ2

```
#define REG_INVERTIQ2 0x3b
```

Definition at line 36 of file [LoRa-RP2040.cpp](#).

8.29.1.35 REG_DIO_MAPPING_1

```
#define REG_DIO_MAPPING_1 0x40
```

Definition at line 37 of file [LoRa-RP2040.cpp](#).

8.29.1.36 REG_VERSION

```
#define REG_VERSION 0x42
```

Definition at line 38 of file [LoRa-RP2040.cpp](#).

8.29.1.37 REG_PA_DAC

```
#define REG_PA_DAC 0x4d
```

Definition at line 39 of file [LoRa-RP2040.cpp](#).

8.29.1.38 MODE_CAD

```
#define MODE_CAD 0x07
```

Definition at line 40 of file [LoRa-RP2040.cpp](#).

8.29.1.39 MODE_LONG_RANGE_MODE

```
#define MODE_LONG_RANGE_MODE 0x80
```

Definition at line 42 of file [LoRa-RP2040.cpp](#).

8.29.1.40 MODE_SLEEP

```
#define MODE_SLEEP 0x00
```

Definition at line 43 of file [LoRa-RP2040.cpp](#).

8.29.1.41 MODE_STDBY

```
#define MODE_STDBY 0x01
```

Definition at line 44 of file [LoRa-RP2040.cpp](#).

8.29.1.42 MODE_TX

```
#define MODE_TX 0x03
```

Definition at line 45 of file [LoRa-RP2040.cpp](#).

8.29.1.43 MODE_RX_CONTINUOUS

```
#define MODE_RX_CONTINUOUS 0x05
```

Definition at line 46 of file [LoRa-RP2040.cpp](#).

8.29.1.44 MODE_RX_SINGLE

```
#define MODE_RX_SINGLE 0x06
```

Definition at line 47 of file [LoRa-RP2040.cpp](#).

8.29.1.45 PA_BOOST

```
#define PA_BOOST 0x80
```

Definition at line 50 of file [LoRa-RP2040.cpp](#).

8.29.1.46 IRQ_TX_DONE_MASK

```
#define IRQ_TX_DONE_MASK 0x08
```

Definition at line 53 of file [LoRa-RP2040.cpp](#).

8.29.1.47 IRQ_PAYLOAD_CRC_ERROR_MASK

```
#define IRQ_PAYLOAD_CRC_ERROR_MASK 0x20
```

Definition at line 54 of file [LoRa-RP2040.cpp](#).

8.29.1.48 IRQ_RX_DONE_MASK

```
#define IRQ_RX_DONE_MASK 0x40
```

Definition at line 55 of file [LoRa-RP2040.cpp](#).

8.29.1.49 RF_MID_BAND_THRESHOLD

```
#define RF_MID_BAND_THRESHOLD 525E6
```

Definition at line 57 of file [LoRa-RP2040.cpp](#).

8.29.1.50 RSSI_OFFSET_HF_PORT

```
#define RSSI_OFFSET_HF_PORT 157
```

Definition at line 58 of file [LoRa-RP2040.cpp](#).

8.29.1.51 RSSI_OFFSET_LF_PORT

```
#define RSSI_OFFSET_LF_PORT 164
```

Definition at line 59 of file [LoRa-RP2040.cpp](#).

8.29.1.52 MAX_PKT_LENGTH

```
#define MAX_PKT_LENGTH 255
```

Definition at line 61 of file [LoRa-RP2040.cpp](#).

8.29.1.53 ISR_PREFIX

```
#define ISR_PREFIX
```

Definition at line 66 of file [LoRa-RP2040.cpp](#).

8.29.2 Variable Documentation

8.29.2.1 LoRa

```
LoRaClass LoRa
```

Definition at line 753 of file [LoRa-RP2040.cpp](#).

8.30 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::parsePacket(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
00258     return packetLength;
00259 }
00260
00261 int LoRaClass::packetRssi()
00262 {
00263     return (readRegister(REG_PKT_RSSI_VALUE) - (_frequency < RF_MID_BAND_THRESHOLD ? RSSI_OFFSET_LF_PORT
00264 : RSSI_OFFSET_HF_PORT));
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.
00285     14)
00286     const float fError = ((static_cast<float>(freqError) * (1L << 24)) / fXtal) * (getSignalBandwidth() /
00287     500000.0f); // p. 37
00288
00289     return static_cast<long>(fError);
00290 }
00291
00292 int LoRaClass::rssи()
00293 {
00294     return (readRegister(REG_RSSI_VALUE) - (_frequency < RF_MID_BAND_THRESHOLD ? RSSI_OFFSET_LF_PORT :
00295 RSSI_OFFSET_HF_PORT));
00296 }
00297
00298 size_t LoRaClass::write(uint8_t byte)
00299 {
00300     return write(&byte, sizeof(byte));
00301 }
00302
00303 size_t LoRaClass::write(const uint8_t *buffer, size_t size)
00304 {
00305     int currentLength = readRegister(REG_PAYLOAD_LENGTH);
00306
00307     // check size
00308     if ((currentLength + size) > MAX_PKT_LENGTH) {
00309         size = MAX_PKT_LENGTH - currentLength;
00310     }
00311
00312     // write data
00313     for (size_t i = 0; i < size; i++) {
00314         writeRegister(REG_FIFO, buffer[i]);
00315     }
00316
00317     // update length
00318     writeRegister(REG_PAYLOAD_LENGTH, currentLength + size);
00319
00320     return size;
00321 }
00322
00323 int LoRaClass::available()
00324 {
00325     return (readRegister(REG_RX_NB_BYTES) - _packetIndex);
00326 }
00327
00328 int LoRaClass::read()
00329 {
00330     if (!available()) {
00331         return -1;
00332     }
00333     _packetIndex++;
00334
00335     return readRegister(REG_FIFO);
00336 }
00337
00338 int LoRaClass::peek()
00339 {
00340     if (!available()) {
00341         return -1;
00342     }
00343
00344     return readRegister(REG_FIFO);
00345 }
```

```

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::setPins(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             } else if ((irqFlags & IRQ_TX_DONE_MASK) != 0) {
00716                 if (_onTxDone) {
00717                     _onTxDone();
00718                 }
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;

```

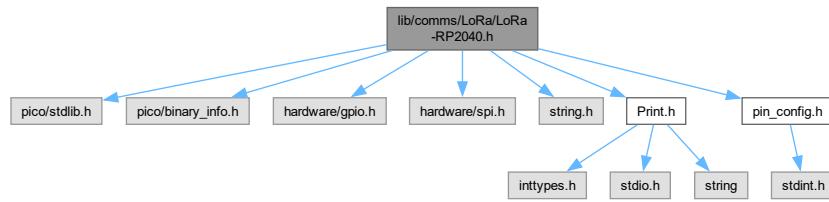
8.31 lib/comms/LoRa/LoRa-RP2040.h File Reference

```

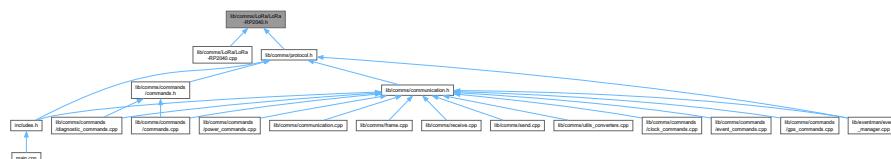
#include "pico/stl.h"
#include "pico/binary_info.h"
#include "hardware/gpio.h"
#include "hardware/spi.h"
#include "string.h"
#include "Print.h"

```

```
#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](#)

8.31.1 Function Documentation

8.31.1.1 [__empty\(\)](#)

```
static void __empty () [static]
```

8.31.2 Variable Documentation

8.31.2.1 [LoRa](#)

```
LoRaClass LoRa [extern]
```

Definition at line [753](#) of file [LoRa-RP2040.cpp](#).

8.32 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 parsePacket(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 setPins(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);
```

```

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;

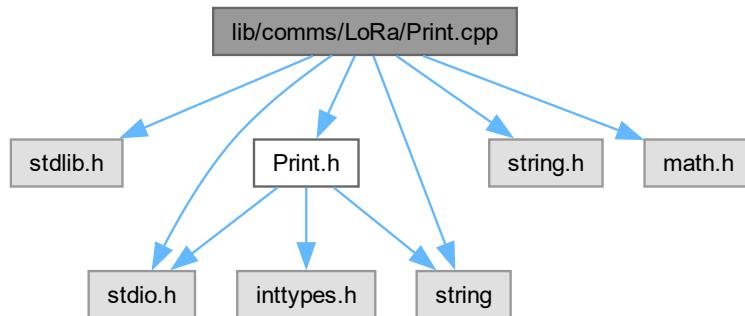
```

8.33 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:

```



8.34 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);
```

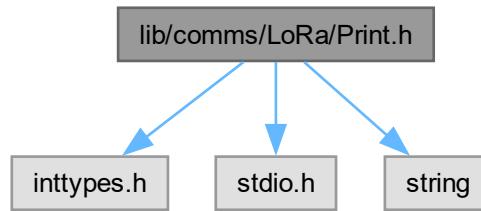
```
00175     n += println();
00176     return n;
00177 }
00178
00179 size_t Print::println(long long num, int base)
00180 {
00181     size_t n = print(num, base);
00182     n += println();
00183     return n;
00184 }
00185
00186 size_t Print::println(unsigned long long num, int base)
00187 {
00188     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;
```

```

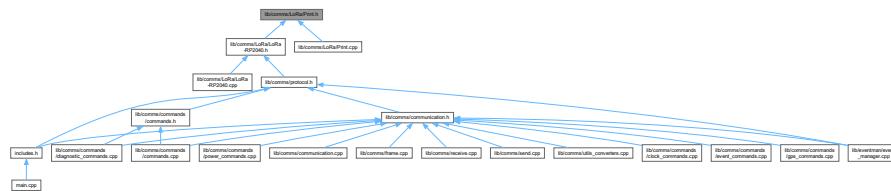
00262     innerLoops++;
00263 }
00264
00265 while (n64 > th16)
00266 {
00267     // 64 bit math part
00268     uint64_t q = n64 / th16;
00269     uint16_t r = n64 - q*th16;
00270     n64 = q;
00271
00272     // 16 bit math loop to do remainder. (note buffer is filled reverse)
00273     for (uint8_t j=0; 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 }
```

8.35 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`

8.35.1 Macro Definition Documentation

8.35.1.1 DEC

```
#define DEC 10
```

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

8.35.1.2 HEX

```
#define HEX 16
```

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

8.35.1.3 OCT

```
#define OCT 8
```

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

8.35.1.4 BIN

```
#define BIN 2
```

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

8.36 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

```

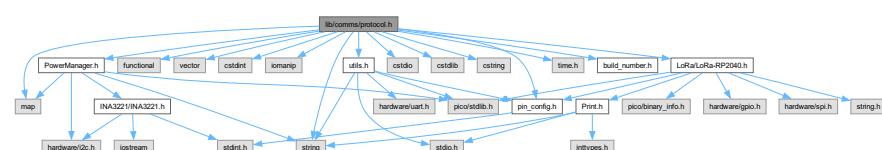
8.37 lib/comms/protocol.h File Reference

```

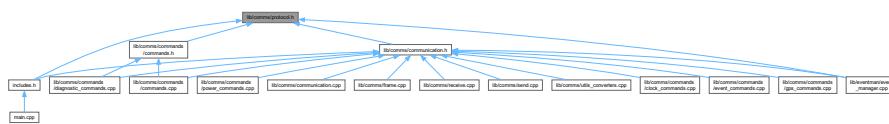
#include <string>
#include <map>
#include <functional>
#include <vector>
#include <cstdint>
#include <iomanip>
#include "pin_config.h"
#include "PowerManager.h"
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include "utils.h"
#include "time.h"
#include "build_number.h"
#include "LoRa/LoRa-RP2040.h"

```

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](#) }

Functions

- std::string [exceptionTypeToString](#) ([ExceptionType](#) type)

Converts an [ExceptionType](#) to a string.
- std::string [operationTypeToString](#) ([OperationType](#) type)

Converts an [OperationType](#) to a string.
- [OperationType](#) [stringToOperationType](#) (const std::string &str)

Converts a string to an [OperationType](#).
- std::vector< uint8_t > [hexStringToBytes](#) (const std::string &hexString)

Converts a hex string to a vector of bytes.
- std::string [valueUnitTypeToString](#) ([ValueUnit](#) unit)

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

Variables

- const std::string [FRAME_BEGIN](#) = "KBST"
- const std::string [FRAME_END](#) = "TSBK"
- const char [DELIMITER](#) = ','

8.37.1 Enumeration Type Documentation

8.37.1.1 ExecutionResult

```
enum class ExecutionResult [strong]
```

Enumerator

| | |
|---------|--|
| SUCCESS | |
| ERROR | |
| INFO | |

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

8.37.1.2 OperationType

```
enum class OperationType [strong]
```

Enumerator

| | |
|-----|--|
| GET | |
| SET | |
| ANS | |
| ERR | |
| INF | |

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

8.37.1.3 CommandAccessLevel

```
enum class CommandAccessLevel [strong]
```

Enumerator

| | |
|------------|--|
| NONE | |
| READ_ONLY | |
| WRITE_ONLY | |
| READ_WRITE | |

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

8.37.1.4 ValueUnit

```
enum class ValueUnit [strong]
```

Enumerator

| | |
|-----------|--|
| UNDEFINED | |
| SECOND | |
| VOLT | |
| BOOL | |
| DATETIME | |
| TEXT | |
| MILIAMP | |

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

8.37.1.5 ExceptionType

```
enum class ExceptionType [strong]
```

Enumerator

| | |
|-------------------|--|
| NONE | |
| NOT_ALLOWED | |
| INVALID_PARAM | |
| INVALID_OPERATION | |
| PARAM_UNNECESSARY | |

Definition at line 57 of file [protocol.h](#).

8.37.2 Function Documentation

8.37.2.1 exceptionTypeToString()

```
std::string exceptionTypeToString (
    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](#).

8.37.2.2 operationTypeToString()

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

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

Parameters

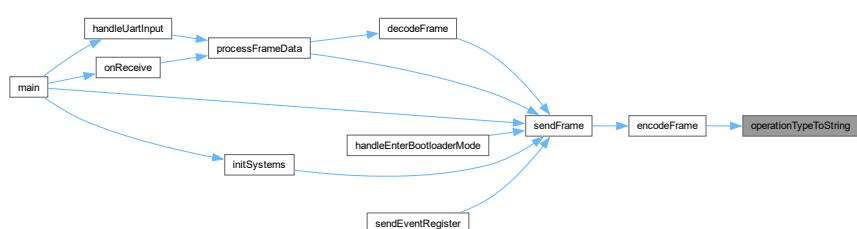
| | |
|-------------|---|
| <i>type</i> | The OperationType to convert. |
|-------------|---|

Returns

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

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

Here is the caller graph for this function:



8.37.2.3 stringToOperationType()

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

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

Parameters

| | |
|------------|------------------------|
| <i>str</i> | 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:



8.37.2.4 hexStringToBytes()

```
std::vector< uint8_t > hexStringToBytes (
    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](#).

8.37.2.5 valueUnitTypeToString()

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

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

Parameters

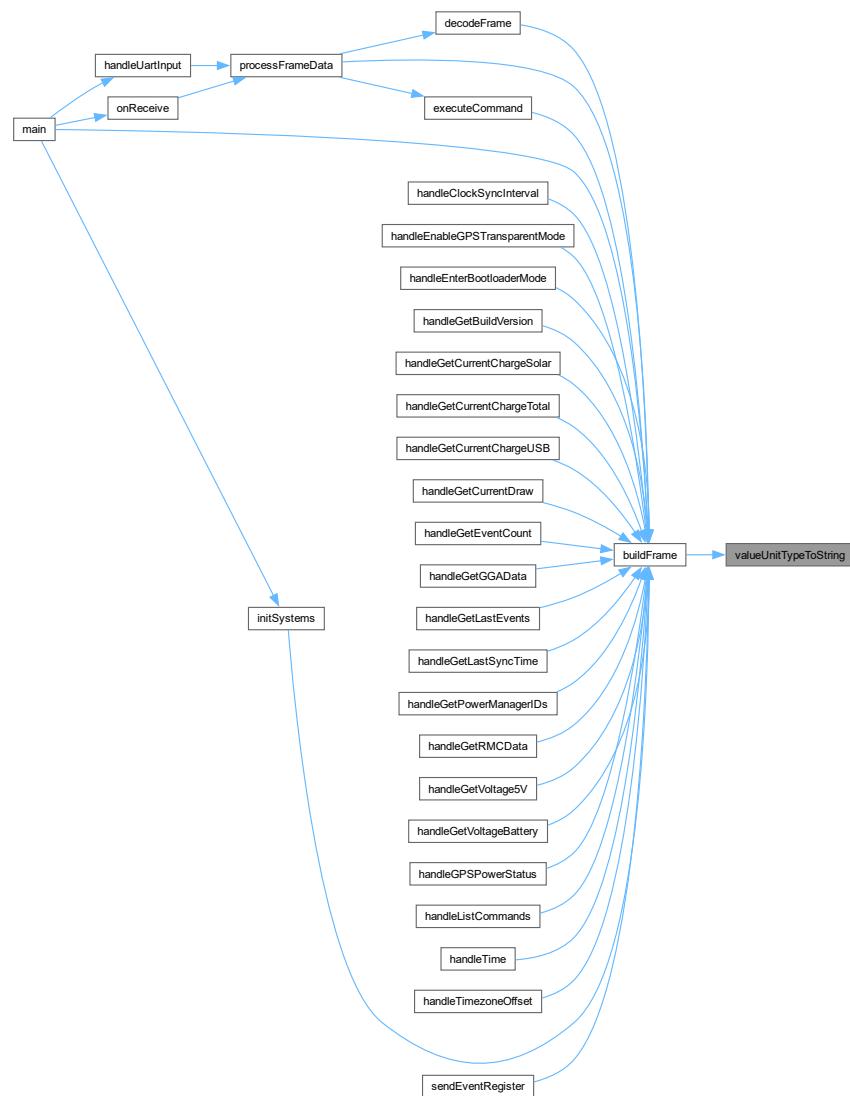
| | |
|-------------|---|
| <i>unit</i> | The ValueUnit to convert. |
|-------------|---|

Returns

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

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

Here is the caller graph for this function:



8.37.3 Variable Documentation

8.37.3.1 FRAME_BEGIN

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

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

8.37.3.2 FRAME_END

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

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

8.37.3.3 DELIMITER

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

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

8.38 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 struct Frame {
00066     std::string header;           // Start marker
00067     uint8_t direction;          // 0 = ground->sat, 1 = sat->ground
00068     OperationType operationType;
00069     uint8_t group;               // Group ID
00070     uint8_t command;             // Command ID within group
00071     std::string value;           // Payload value
00072     std::string unit;            // Payload unit
00073     std::string footer;          // End marker
00074 };
00075
00076 std::string exceptionTypeToString(ExceptionType type);
00077 std::string operationTypeToString(OperationType type);
00078 OperationType stringToOperationType(const std::string& str);
00079 std::vector<uint8_t> hexStringToBytes(const std::string& hexString);
00080 std::string valueUnitTypeToString(ValueUnit unit);
00081
00082 #endif

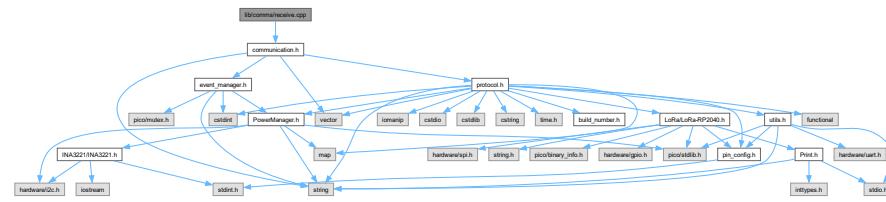
```

8.39 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 [onReceive](#) (int packetSize)
Callback function for handling received LoRa packets.
- void [handleUartInput](#) ()
Handles UART input.

8.39.1 Detailed Description

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

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

8.39.2 Function Documentation

8.39.2.1 onReceive()

```
void onReceive (
    int packetSize)
```

Callback function for handling received LoRa packets.

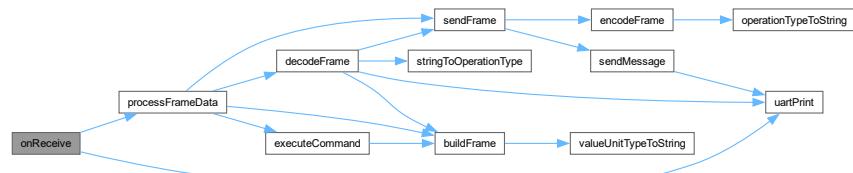
Parameters

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

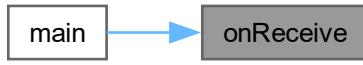
Reads the received LoRa packet, extracts metadata, validates the destination and local addresses, extracts the frame data, and processes it. Prints raw hex values for debugging.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.39.2.2 handleUartInput()

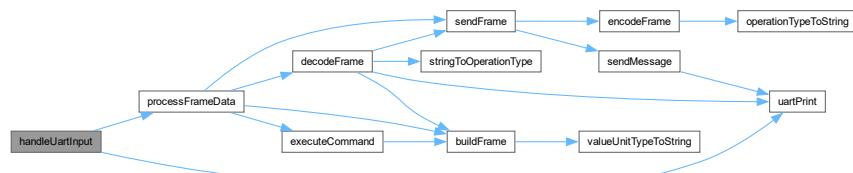
```
void handleUartInput ()
```

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 77 of file [receive.cpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



8.40 receive.cpp

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

```

00001 #include "communication.h"
00002
00003
00008
00015 void onReceive(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 != localAddress) {
00031         uartPrint("Error: Destination address mismatch!");
00032         return;
00033     }
00034
00035     if (receivedLocalAddress != destination) {
00036         uartPrint("Error: Local address mismatch!");
00037         return;
00038     }
00039
00040     // Find the starting index of the actual frame data
00041     int startIndex = 2; // Start after the metadata
00042
00043     // Extract the frame data
00044     std::string received = std::string(reinterpret_cast<char*>(buffer + startIndex), bytesRead -
00045     startIndex);
00046     if (received.empty()) return;
00047
00048     // Debug: Print raw hex values
00049     std::stringstream hexDump;
00050     hexDump << "Raw bytes: ";
00051     for (int i = 0; i < bytesRead; i++) {
00052         hexDump << std::hex << std::setfill('0') << std::setw(2)
00053             << static_cast<int>(buffer[i]) << " ";
00054     }
00055     uartPrint(hexDump.str());
00056
00057     // Find frame boundaries
00058     size_t headerPos = received.find(FRAME_BEGIN);
00059     size_t footerPos = received.find(FRAME_END);
00060
00061     if (headerPos != std::string::npos && footerPos != std::string::npos && footerPos > headerPos) {
00062         // Extract frame between header and footer
00063         std::string frameData = received.substr(headerPos, footerPos + FRAME_END.length() -
00064             headerPos);
00064         uartPrint("Extracted frame (length=" + std::to_string(frameData.length()) + "): " +
00065             frameData);
00065         processFrameData(frameData);
00066     } else {
00067         uartPrint("No valid frame found in received data");

```

```

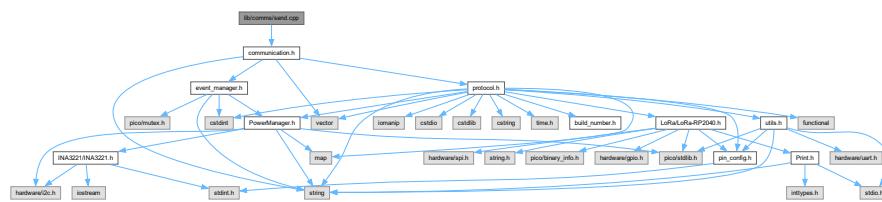
00068      }
00069  }
00070
00071
00077 void handleUartInput() {
00078     static std::string uartBuffer; // Static buffer to store UART input
00079
00080     while (uart_is_readable(DEBUG_UART_PORT)) {
00081         char c = uart_getc(DEBUG_UART_PORT);
00082
00083         if (c == '\r' || c == '\n') {
00084             uartPrint("Received UART string: " + uartBuffer);
00085             processFrameData(uartBuffer); // Process the data
00086             uartBuffer.clear(); // Clear the buffer for the next input
00087         } else {
00088             // Append the character to the buffer
00089             uartBuffer += c;
00090         }
00091     }
00092 }
```

8.41 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 [sendMessage \(string outgoing\)](#)
Sends a message using LoRa.
- void [sendFrame \(const Frame &frame\)](#)
Sends a [Frame](#) using LoRa.
- void [sendLargePacket \(const uint8_t *data, size_t length\)](#)
Sends a large packet using LoRa.

8.41.1 Detailed Description

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

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

8.41.2 Function Documentation

8.41.2.1 sendMessage()

```
void sendMessage (
    string outgoing)
```

Sends a message using LoRa.

Parameters

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

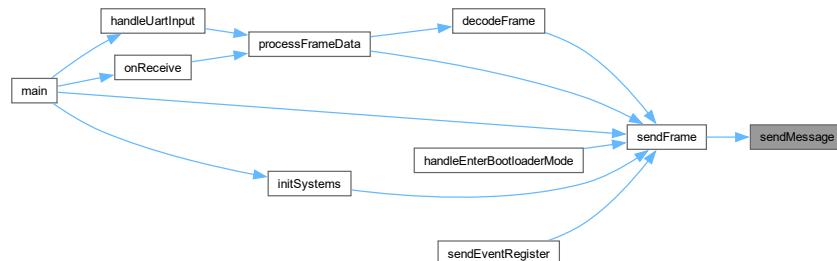
Converts the outgoing string to a C-style string, adds destination and local addresses, and sends the message using LoRa. Prints a log message to the UART.

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

Here is the call graph for this function:



Here is the caller graph for this function:

**8.41.2.2 sendFrame()**

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

Sends a [Frame](#) using LoRa.

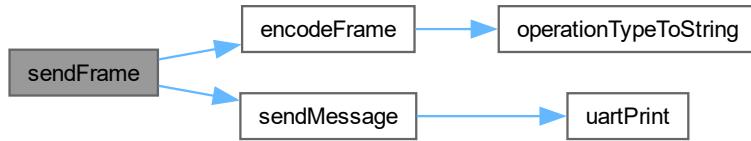
Parameters

| | |
|--------------|------------------------------------|
| <i>frame</i> | The Frame to send. |
|--------------|------------------------------------|

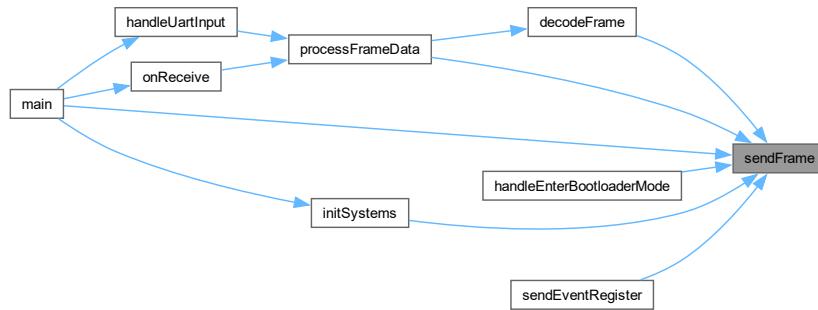
Encodes the [Frame](#) into a string and sends it using the [sendMessage](#) function.

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

Here is the call graph for this function:



Here is the caller graph for this function:



8.41.2.3 sendLargePacket()

```
void sendLargePacket (
    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 55 of file [send.cpp](#).

8.42 send.cpp

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

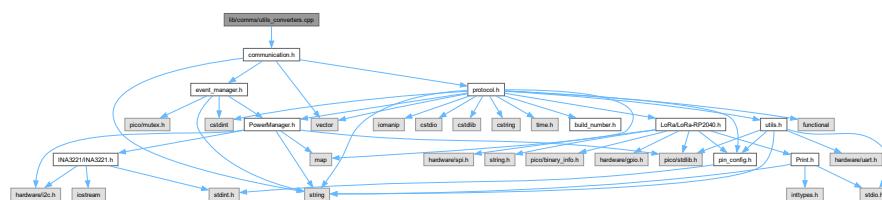
```

00001 #include "communication.h"
00002
00003
00008
00015 void sendMessage(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(destination);    // add destination address
00023     LoRa.write(localAddress);   // 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(destination);
00029     messageToLog += " containing: " + string(send);
00030
00031     uartPrint(messageToLog);
00032
00033     LoRa.flush();
00034 }
00035
00036
00042 void sendFrame(const Frame& frame) {
00043     std::string encodedFrame = encodeFrame(frame);
00044     // sendLargePacket(data, encodedFrame);
00045     sendMessage(encodedFrame);
00046 }
00047
00048
00055 void sendLargePacket(const uint8_t* data, size_t length)
00056 {
00057     const size_t MAX_PKT_SIZE = 255;
00058     size_t offset = 0;
00059     while (offset < length)
00060     {
00061         size_t chunkSize = ((length - offset) < MAX_PKT_SIZE) ? (length - offset) : MAX_PKT_SIZE;
00062         LoRa.beginPacket();
00063         LoRa.write(&data[offset], chunkSize);
00064         LoRa.endPacket();
00065         offset += chunkSize;
00066         sleep_ms(100);
00067     }
00068 }
```

8.43 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 [exceptionTypeToString](#) ([ExceptionType](#) type)
Converts an [ExceptionType](#) to a string.
- std::string [valueUnitTypeToString](#) ([ValueUnit](#) unit)
Converts a [ValueUnit](#) to a string.
- std::string [operationTypeToString](#) ([OperationType](#) type)
Converts an [OperationType](#) to a string.
- [OperationType](#) [stringToOperationType](#) (const std::string &str)
Converts a string to an [OperationType](#).
- std::vector< uint8_t > [hexStringToBytes](#) (const std::string &hexString)
Converts a hex string to a vector of bytes.

8.43.1 Detailed Description

Implements utility functions for converting between different data types.

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

8.43.2 Function Documentation

8.43.2.1 exceptionTypeToString()

```
std::string exceptionTypeToString (
    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](#).

8.43.2.2 valueUnitTypeToString()

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

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

Parameters

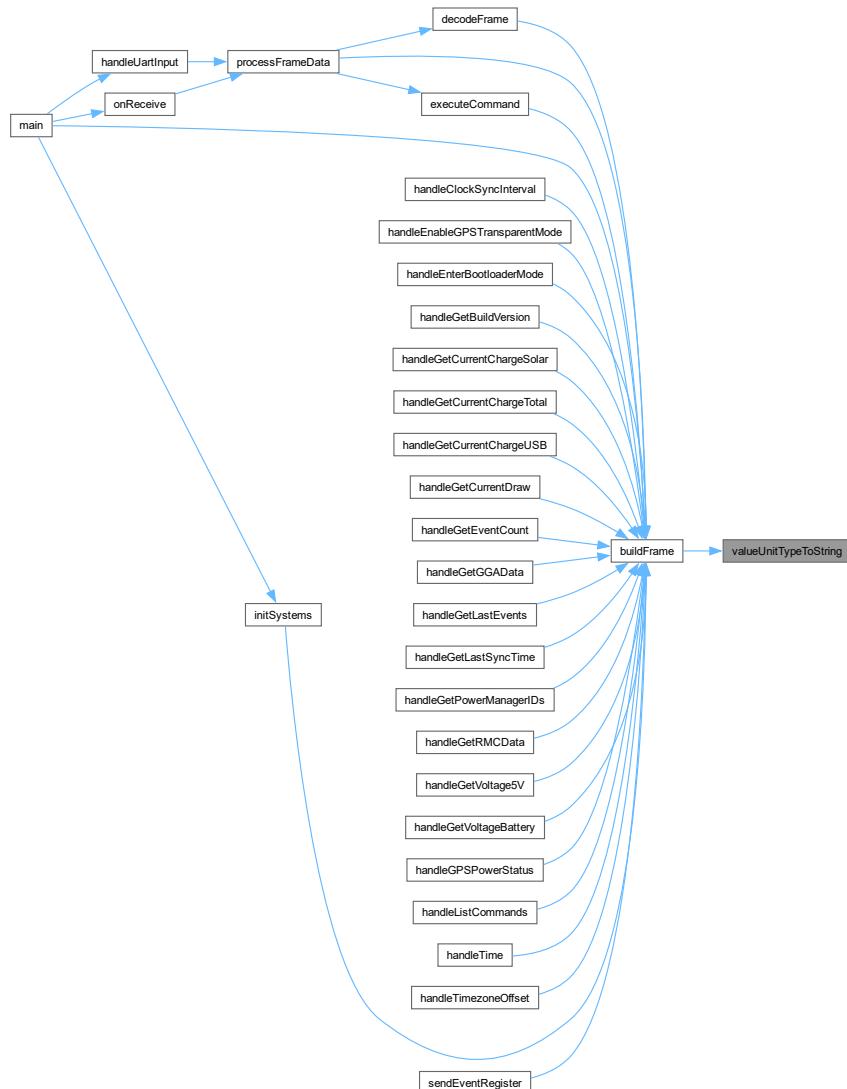
| | |
|-------------|---|
| <i>unit</i> | The ValueUnit to convert. |
|-------------|---|

Returns

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

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

Here is the caller graph for this function:



8.43.2.3 operationTypeToString()

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

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

Parameters

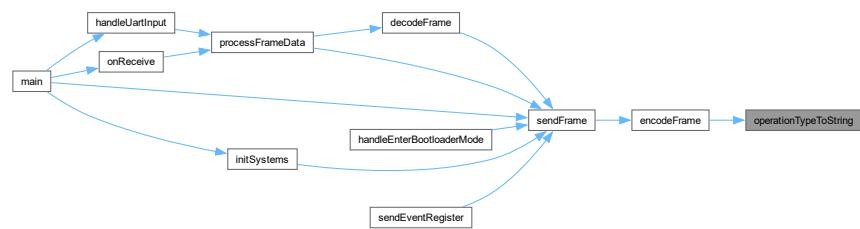
| | |
|-------------------|---|
| <code>type</code> | The OperationType to convert. |
|-------------------|---|

Returns

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

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

Here is the caller graph for this function:



8.43.2.4 stringToOperationType()

```
OperationType stringToOperationType (
    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:



8.43.2.5 hexStringToBytes()

```
std::vector< uint8_t > hexStringToBytes (
    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](#).

8.44 utils_converters.cpp

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

```
00001 #include "communication.h"
00002
00003
00008
00014 std::string exceptionTypeToString(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 valueUnitTypeToString(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 operationTypeToString(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 stringToOperationType(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 }
```

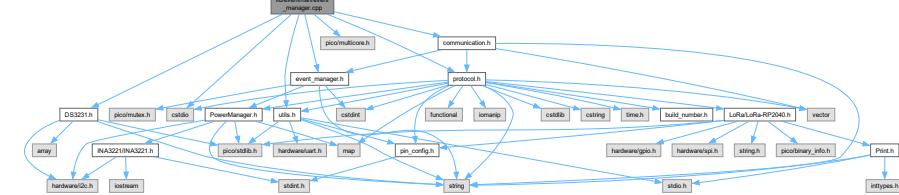
```
00075
00081 std::vector<uint8_t> hexStringToBytes(const std::string& hexString) {
00082     std::vector<uint8_t> bytes;
00083     for (size_t i = 0; i < hexString.length(); i += 2) {
00084         std::string byteString = hexString.substr(i, 2);
00085         unsigned int byte;
00086         std::stringstream ss;
00087         ss << std::hex << byteString;
00088         ss >> byte;
00089         bytes.push_back(static_cast<uint8_t>(byte));
00090     }
00091     return bytes;
00092 }
```

8.45 lib/eventman/event_manager.cpp File Reference

Implements the event management system for the Kubit firmware.

```
#include "event_manager.h"
#include <cstdio>
#include "protocol.h"
#include "pico/multicore.h"
#include "communication.h"
#include "utils.h"
#include "DS3231.h"
Include dependency graph for event_manager.cpp:
```

www.merriam-webster.com



Functions

- void **checkPowerEvents** (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

- static int **fallingTrendCount** = 0
Voltage threshold for detecting an overcharge condition.
- 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.

8.45.1 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](#).

8.45.2 Function Documentation

8.45.2.1 checkPowerEvents()

```
void checkPowerEvents (
    PowerManager & pm)
```

Checks power statuses and triggers events based on voltage trends.

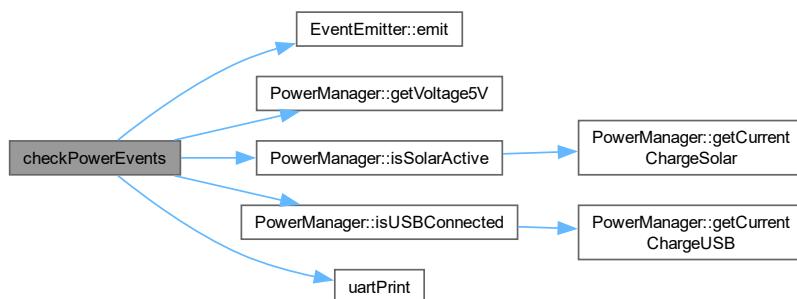
Parameters

| | |
|-----------|---|
| <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:



8.45.3 Variable Documentation

8.45.3.1 eventLogId

```
volatile uint16_t eventLogId = 0
```

Global event log ID counter.

Definition at line [20](#) of file [event_manager.cpp](#).

8.45.3.2 lastPowerState

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

Stores the last known power state.

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

8.45.3.3 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](#).

8.45.3.4 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](#).

8.45.3.5 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](#).

8.45.3.6 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](#).

8.45.3.7 fallingTrendCount

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

Counter for consecutive falling voltage readings.

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

8.45.3.8 lastSolarState

```
bool lastSolarState = false
```

Stores the last known solar charging state.

Definition at line [55](#) of file [event_manager.cpp](#).

8.45.3.9 lastUSBState

```
bool lastUSBState = false
```

Stores the last known USB connection state.

Definition at line [60](#) of file [event_manager.cpp](#).

8.45.3.10 systemClock

```
DS3231 systemClock [extern]
```

External declaration of the system clock.

8.45.3.11 eventManager

EventManagerImpl eventManager

Global instance of the `EventManager` implementation.

Global instance of the `EventManagerImpl` class.

Definition at line 70 of file `event_manager.cpp`.

8.46 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::logEvent(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.getTimeUnix();
00086     log.group = group;
00087     log.event = event;
00088
00089     // Print event immediately
00090     uartPrint(log.toString());
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         saveToStorage();
00103     }
00104
00105     mutex_exit(&eventMutex);
00106 }
00107
00108
00114 const EventLog& EventManager::getEvent(size_t index) const {
```

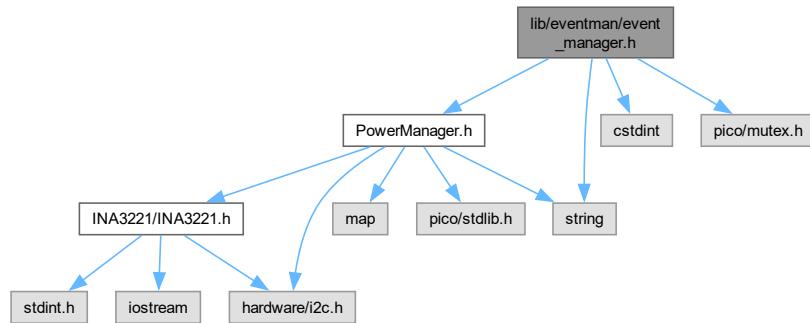
```

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
00132
00133 void checkPowerEvents(PowerManager& pm) {
00134     float currentVoltage = pm.getVoltage5V();
00135     static float previousVoltage = 0.0f;
00136     float delta = currentVoltage - previousVoltage;
00137     previousVoltage = currentVoltage;
00138
00139     if (delta < FALL_RATE_THRESHOLD) {
00140         fallingTrendCount++;
00141     } else {
00142         fallingTrendCount = 0;
00143     }
00144
00145     if (fallingTrendCount >= FALLING_TREND_REQUIRED) {
00146         uartPrint("Power falling detected!");
00147         lastPowerState = PowerEvent::POWER_FALLING;
00148         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_FALLING);
00149     }
00150
00151     if (currentVoltage < PowerManager::VOLTAGE_LOW_THRESHOLD &&
00152         lastPowerState != PowerEvent::LOW_BATTERY) {
00153         uartPrint("Low battery detected!");
00154         lastPowerState = PowerEvent::LOW_BATTERY;
00155         EventEmitter::emit(EventGroup::POWER, PowerEvent::LOW_BATTERY);
00156     }
00157     else if (currentVoltage > PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&
00158             lastPowerState != PowerEvent::OVERCHARGE) {
00159         uartPrint("Overcharge detected!");
00160         lastPowerState = PowerEvent::OVERCHARGE;
00161         EventEmitter::emit(EventGroup::POWER, PowerEvent::OVERCHARGE);
00162     }
00163     else if (currentVoltage >= PowerManager::VOLTAGE_LOW_THRESHOLD &&
00164             currentVoltage <= PowerManager::VOLTAGE_OVERCHARGE_THRESHOLD &&
00165             lastPowerState != PowerEvent::POWER_NORMAL) {
00166         uartPrint("Power back to normal!");
00167         lastPowerState = PowerEvent::POWER_NORMAL;
00168         EventEmitter::emit(EventGroup::POWER, PowerEvent::POWER_NORMAL);
00169     }
00170
00171     // Check solar charging state
00172     bool currentSolarState = pm.isSolarActive();
00173     if (currentSolarState != lastSolarState) {
00174         if (currentSolarState) {
00175             uartPrint("Solar charging active!");
00176             EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_ACTIVE);
00177         } else {
00178             uartPrint("Solar charging inactive!");
00179             EventEmitter::emit(EventGroup::POWER, PowerEvent::SOLAR_INACTIVE);
00180         }
00181         lastSolarState = currentSolarState;
00182     }
00183
00184
00185
00186
00187
00188
00189
00190     // Check USB connection state
00191     bool currentUSBState = pm.isUSBConnected();
00192     if (currentUSBState != lastUSBState) {
00193         if (currentUSBState) {
00194             uartPrint("USB connected!");
00195             EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_CONNECTED);
00196         } else {
00197             uartPrint("USB disconnected!");
00198             EventEmitter::emit(EventGroup::POWER, PowerEvent::USB_DISCONNECTED);
00199         }
00200         lastUSBState = currentUSBState;
00201     }
00202 }

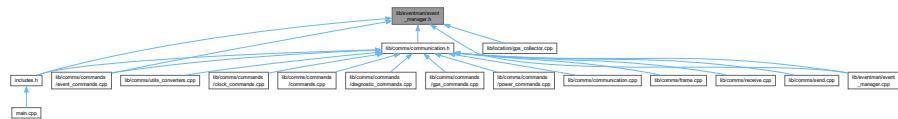
```

8.47 lib/eventman/event_manager.h File Reference

```
#include "PowerManager.h"
#include <cstdint>
#include <string>
#include "pico/mutex.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 1000`

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 `toString () const`

Converts the `EventLog` to a string representation.
- void `checkPowerEvents (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.

8.47.1 Macro Definition Documentation

8.47.1.1 EVENT_BUFFER_SIZE

```
#define EVENT_BUFFER_SIZE 1000
```

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

8.47.2 Enumeration Type Documentation

8.47.2.1 EventGroup

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

Enumerator

| | |
|--------|--|
| SYSTEM | |
| POWER | |
| COMMS | |
| GPS | |
| CLOCK | |

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

8.47.2.2 SystemEvent

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

Enumerator

| | |
|----------------|--|
| BOOT | |
| SHUTDOWN | |
| WATCHDOG_RESET | |
| CORE1_START | |
| CORE1_STOP | |

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

8.47.2.3 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 30 of file [event_manager.h](#).

8.47.2.4 CommsEvent

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

Enumerator

| | |
|--------------|--|
| RADIO_INIT | |
| RADIO_ERROR | |
| MSG_RECEIVED | |
| MSG_SENT | |
| UART_ERROR | |

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

8.47.2.5 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 51 of file [event_manager.h](#).

8.47.2.6 ClockEvent

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

Enumerator

| | |
|----------|--|
| CHANGED | |
| GPS_SYNC | |

Definition at line 63 of file [event_manager.h](#).

8.47.3 Function Documentation**8.47.3.1 __attribute__()**

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

8.47.3.2 `toString()`

```
std::string __attribute__::toString () const
```

Converts the [EventLog](#) to a string representation.

Returns

A string representation of the [EventLog](#).

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

8.47.3.3 `checkPowerEvents()`

```
void checkPowerEvents (
    PowerManager & pm)
```

Checks power statuses and triggers events based on voltage trends.

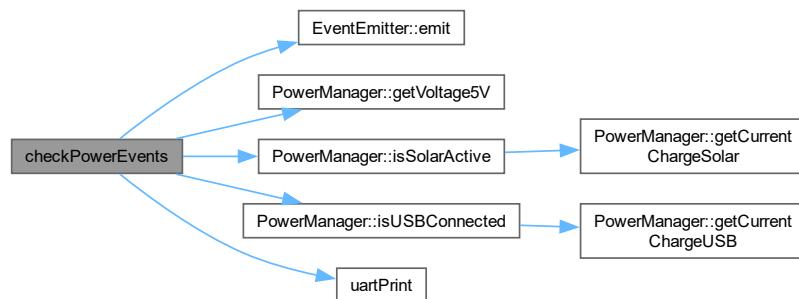
Parameters

| | |
|-----------|---|
| <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:



8.47.4 Variable Documentation

8.47.4.1 id

```
uint16_t id
```

Sequence number.

Definition at line 1 of file [event_manager.h](#).

8.47.4.2 timestamp

```
uint32_t timestamp
```

Unix timestamp or system time.

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

8.47.4.3 group

```
uint8_t group
```

Event group identifier.

Definition at line 3 of file [event_manager.h](#).

8.47.4.4 event

```
uint8_t event
```

Specific event identifier.

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

8.47.4.5 __attribute__

```
class EventManager __attribute__
```

8.47.4.6 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](#).

8.48 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
00009 #define EVENT_BUFFER_SIZE 1000
0010
0011 // Event Groups
0012 enum class EventGroup : uint8_t {
0013     SYSTEM    = 0x00,
0014     POWER     = 0x01,
0015     COMMS     = 0x02,
0016     GPS        = 0x03,
0017     CLOCK      = 0x04
0018 };
0019
0020 // System Events
0021 enum class SystemEvent : uint8_t {
0022     BOOT       = 0x01,
0023     SHUTDOWN   = 0x02,
0024     WATCHDOG_RESET = 0x03,
0025     CORE1_START = 0x04,
0026     CORE1_STOP  = 0x05
0027 };
0028
0029 // Power Events
0030 enum class PowerEvent : uint8_t {
0031     LOW_BATTERY    = 0x01,
0032     OVERCHARGE    = 0x02,
0033     POWER_FALLING = 0x03,
0034     POWER_NORMAL   = 0x04,
0035     SOLAR_ACTIVE   = 0x05,
0036     SOLAR_INACTIVE = 0x06,
0037     USB_CONNECTED  = 0x07,
0038     USB_DISCONNECTED = 0x08
0039 };
0040
0041 // Communication Events
0042 enum class CommsEvent : uint8_t {
0043     RADIO_INIT    = 0x01,
0044     RADIO_ERROR   = 0x02,
0045     MSG_RECEIVED  = 0x03,
0046     MSG_SENT      = 0x04,
0047     UART_ERROR    = 0x06
0048 };
0049
0050 // GPS Events
0051 enum class GPSEvent : uint8_t {
0052     LOCK          = 0x01,
0053     LOST          = 0x02,
0054     ERROR         = 0x03,
0055     POWER_ON      = 0x04,
0056     POWER_OFF     = 0x05,
0057     DATA_READY    = 0x06,
0058     PASS_THROUGH_START = 0x07,
0059     PASS_THROUGH_END = 0x08,
0060 };
0061
0062 // Clock Events
0063 enum class ClockEvent : uint8_t {
0064     CHANGED       = 0x01,
0065     GPS_SYNC      = 0x02
0066 };
0067
0068
0073 class EventLog {
0074 public:
0075     uint16_t id;
0076     uint32_t timestamp;
0077     uint8_t group;
0078     uint8_t event;
0079
0084     std::string toString() const {
0085         char buffer[256];
0086         snprintf(buffer, sizeof(buffer),
0087                 "EventLog: id=%u, timestamp=%lu, group=%u, event=%u",
0088                 id, timestamp, group, event);
0089         return std::string(buffer);
0090     }

```

```

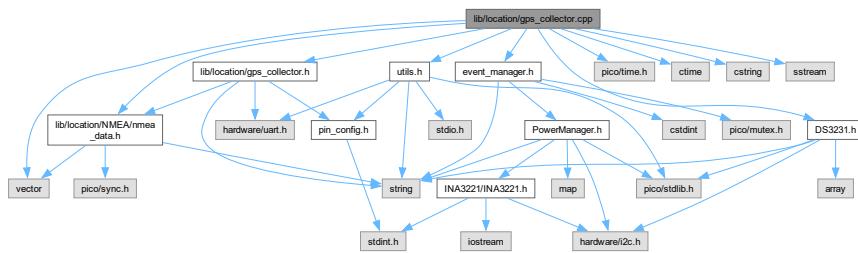
00091 } __attribute__((packed));
00092
00093
00098 class EventManager {
00099 public:
00104     EventManager()
00105         : eventCount(0)
00106         , writeIndex(0)
00107         , nextEventId(0)
00108         , needsPersistence(false)
00109     {
00110         mutex_init(&eventMutex);
00111     }
00112
00116     virtual ~EventManager() = default;
00117
00122     virtual void init() {
00123         loadFromStorage();
00124     }
00125
00131     void logEvent(uint8_t group, uint8_t event);
00132
00138     const EventLog& getEvent(size_t index) const;
00139
00144     size_t getCount() const { return eventCount; }
00145
00150     virtual bool saveToStorage() = 0;
00151
00156     virtual bool loadFromStorage() = 0;
00157
00158 protected:
00159     EventLog events[EVENT_BUFFER_SIZE];
00160     size_t eventCount;
00161     size_t writeIndex;
00162     mutex_t eventMutex;
00163     volatile uint16_t nextEventId;
00164     bool needsPersistence;
00165 };
00166
00167
00172 class EventManagerImpl : public EventManager {
00173 public:
00178     EventManagerImpl() {
00179         init(); // Safe to call virtual functions here
00180     }
00181
00187     bool saveToStorage() override {
00188         // TODO: Implement based on chosen storage (SD/EEPROM)
00189         needsPersistence = false;
00190         return true;
00191     }
00192
00198     bool loadFromStorage() override {
00199         // TODO: Implement based on chosen storage (SD/EEPROM)
00200         return false;
00201     }
00202 };
00203
00204
00208 extern EventManagerImpl eventManager;
00209
00214 class EventEmitter {
00215 public:
00222     template<typename T>
00223     static void emit(EventGroup group, T event) {
00224         eventManager.logEvent(
00225             static_cast<uint8_t>(group),
00226             static_cast<uint8_t>(event)
00227         );
00228     }
00229 };
00230
00231
00236 void checkPowerEvents(PowerManager& pm);
00237
00238 #endif

```

8.49 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 collectGPSData ()`

Variables

- `NMEAData nmea_data`

8.49.1 Macro Definition Documentation

8.49.1.1 MAX_RAW_DATA_LENGTH

```
#define MAX_RAW_DATA_LENGTH 1024
```

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

8.49.2 Function Documentation

8.49.2.1 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:

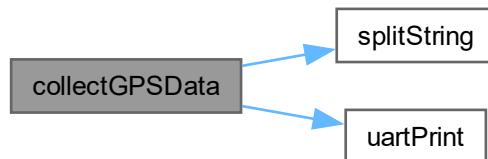


8.49.2.2 collectGPSData()

```
void collectGPSData ()
```

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:



8.49.3 Variable Documentation

8.49.3.1 nmea_data

NMEAData nmea_data [extern]

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

8.50 gps_collector.cpp

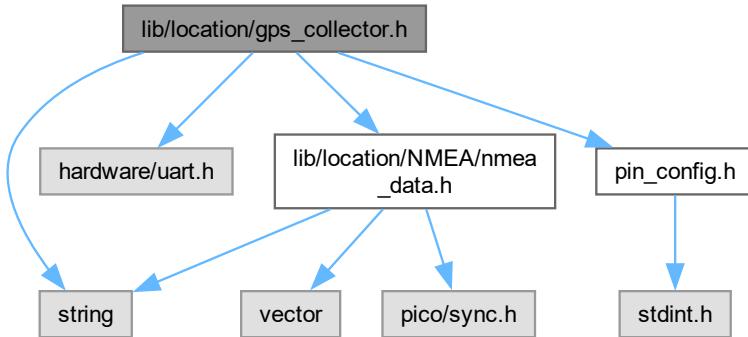
[Go to the documentation of this file.](#)

```

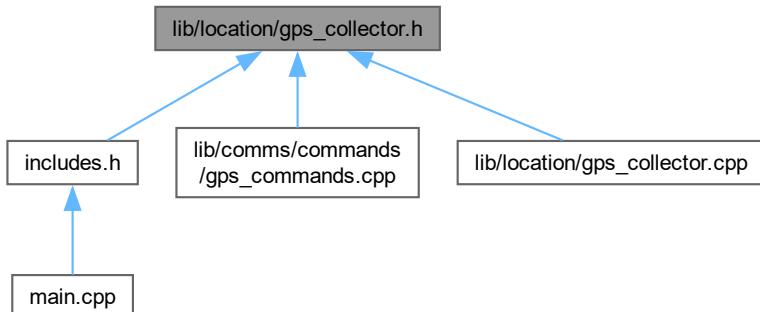
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/gps_collector.cpp
00002 #include "lib/location/gps_collector.h"
00003 #include "utils.h"
00004 #include "pico/time.h"
00005 #include "lib/location/NMEA/nmea_data.h"
00006 #include "event_manager.h"
00007 #include <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 collectGPSdata() {
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("$GPRMC") == 0) {
00046                     nmea_data.updateRmcTokens(tokens);
00047                     uartPrint("RMC data received!");
00048                     uartPrint(message.c_str());
00049                 } else if (message.find("$GPGGA") == 0) {
00050                     nmea_data.updateGgaTokens(tokens);
00051                     uartPrint("GGA data received!");
00052                     uartPrint(message.c_str());
00053                 }
00054             }
00055         } else {
00056             // Append to buffer
00057             if (raw_data_index < MAX_RAW_DATA_LENGTH - 1) {
00058                 raw_data_buffer[raw_data_index++] = c;
00059             } else {
00060                 uartPrint("GPS data overflow!");
00061                 raw_data_index = 0;
00062             }
00063         }
00064     }
00065 }
```

8.51 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 [collectGPSData \(\)](#)

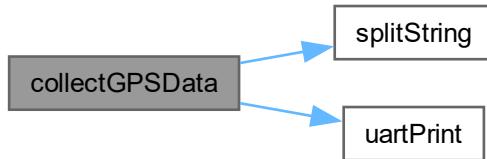
8.51.1 Function Documentation

8.51.1.1 collectGPSData()

```
void collectGPSData ()
```

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:



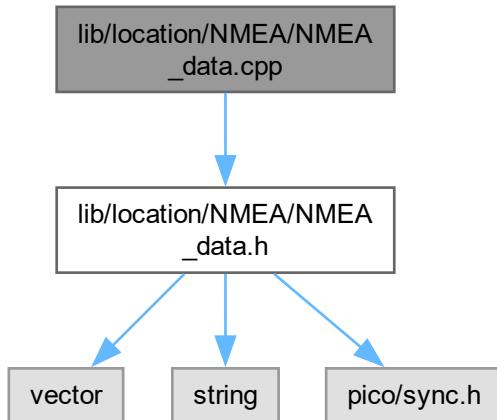
8.52 gps_collector.h

[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 collectGPSData();
00011
00012 #endif
```

8.53 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](#)

8.53.1 Variable Documentation

8.53.1.1 nmea_data

[NMEAData nmea_data](#)

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

8.54 NMEA_data.cpp

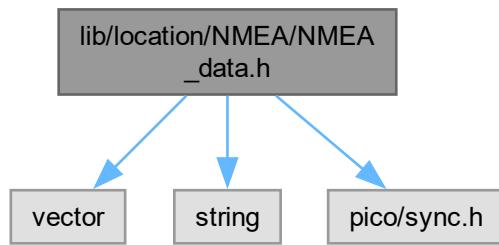
[Go to the documentation of this file.](#)

```
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::updateRmcTokens(const std::vector<std::string>& tokens) {
00011     mutex_enter_blocking(&rmc_mutex);
00012     rmcTokens = tokens;
00013     mutex_exit(&rmc_mutex);
00014 }
```

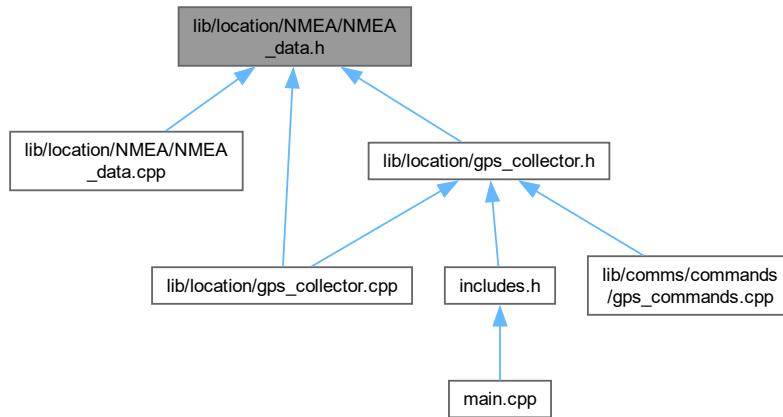
```
00015
00016 void NMEAData::updateGgaTokens(const std::vector<std::string>& tokens) {
00017     mutex_enter_blocking(&gga_mutex);
00018     ggaTokens = tokens;
00019     mutex_exit(&gga_mutex);
00020 }
00021
00022 std::vector<std::string> NMEAData::getRmcTokens() const {
00023     mutex_enter_blocking(const_cast<mutex_t*>(&rmc_mutex));
00024     std::vector<std::string> copy = rmcTokens;
00025     mutex_exit(const_cast<mutex_t*>(&rmc_mutex));
00026     return copy;
00027 }
00028
00029 std::vector<std::string> NMEAData::getGgaTokens() const {
00030     mutex_enter_blocking(const_cast<mutex_t*>(&gga_mutex));
00031     std::vector<std::string> copy = ggaTokens;
00032     mutex_exit(const_cast<mutex_t*>(&gga_mutex));
00033     return copy;
00034 }
```

8.55 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](#)

8.55.1 Variable Documentation

8.55.1.1 nmea_data

`NMEAData nmea_data [extern]`

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

8.56 NMEA_data.h

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

```

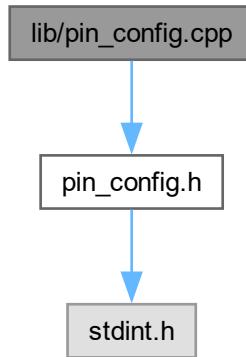
00001 // filepath: /c:/Users/Kuba/Desktop/inz/kubisat/software/kubisat_firmware/lib/GPS/nmea_data.h
00002 #ifndef NMEA_DATA_H
00003 #define NMEA_DATA_H
00004
00005 #include <vector>
00006 #include <string>
00007 #include "pico/sync.h"
00008
00009 class NMEAData {
00010 public:
00011     NMEAData();
00012     void updateRmcTokens(const std::vector<std::string>& tokens);
00013     void updateGgaTokens(const std::vector<std::string>& tokens);
00014     std::vector<std::string> getRmcTokens() const;
00015 }
```

```
00016     std::vector<std::string> getGgaTokens() const;
00017
00018 private:
00019     std::vector<std::string> rmcTokens;
00020     std::vector<std::string> ggaTokens;
00021     mutex_t rmc_mutex;
00022     mutex_t gga_mutex;
00023 };
00024
00025 extern NMEAData nmea_data;
00026
00027 #endif
```

8.57 lib/pin_config.cpp File Reference

```
#include "pin_config.h"
```

Include dependency graph for pin_config.cpp:



Variables

- const int `csPin` = 17
- const int `resetPin` = 22
- const int `irqPin` = 28
- uint8_t `localAddress` = 37
- uint8_t `destination` = 21

8.57.1 Variable Documentation

8.57.1.1 csPin

```
const int csPin = 17
```

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

8.57.1.2 resetPin

```
const int resetPin = 22
```

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

8.57.1.3 irqPin

```
const int irqPin = 28
```

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

8.57.1.4 localAddress

```
uint8_t localAddress = 37
```

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

8.57.1.5 destination

```
uint8_t destination = 21
```

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

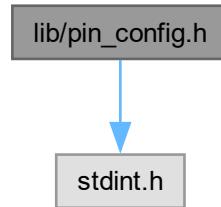
8.58 pin_config.cpp

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

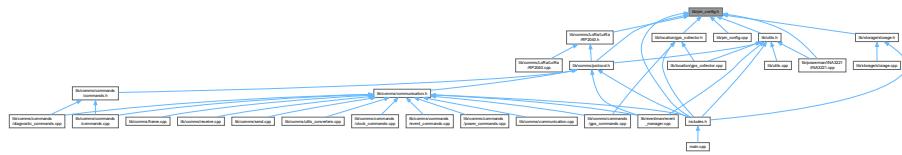
```
00001 #include "pin_config.h"
00002
00003 // LoRa constants
00004 const int csPin = 17;           // LoRa radio chip select
00005 const int resetPin = 22;        // LoRa radio reset
00006 const int irqPin = 28;          // LoRa hardware interrupt pin
00007
00008 uint8_t localAddress = 37;      //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

8.59 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 csPin
- const int resetPin
- const int irqPin
- uint8_t localAddress
- uint8_t destination

8.59.1 Macro Definition Documentation

8.59.1.1 DEBUG_UART_PORT

```
#define DEBUG_UART_PORT uart0
```

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

8.59.1.2 DEBUG_UART_BAUD_RATE

```
#define DEBUG_UART_BAUD_RATE 115200
```

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

8.59.1.3 DEBUG_UART_TX_PIN

```
#define DEBUG_UART_TX_PIN 0
```

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

8.59.1.4 DEBUG_UART_RX_PIN

```
#define DEBUG_UART_RX_PIN 1
```

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

8.59.1.5 MAIN_I2C_PORT

```
#define MAIN_I2C_PORT i2c1
```

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

8.59.1.6 MAIN_I2C_SDA_PIN

```
#define MAIN_I2C_SDA_PIN 6
```

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

8.59.1.7 MAIN_I2C_SCL_PIN

```
#define MAIN_I2C_SCL_PIN 7
```

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

8.59.1.8 GPS_UART_PORT

```
#define GPS_UART_PORT uart1
```

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

8.59.1.9 GPS_UART_BAUD_RATE

```
#define GPS_UART_BAUD_RATE 9600
```

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

8.59.1.10 GPS_UART_TX_PIN

```
#define GPS_UART_TX_PIN 8
```

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

8.59.1.11 GPS_UART_RX_PIN

```
#define GPS_UART_RX_PIN 9
```

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

8.59.1.12 GPS_POWER_ENABLE_PIN

```
#define GPS_POWER_ENABLE_PIN 14
```

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

8.59.1.13 BUFFER_SIZE

```
#define BUFFER_SIZE 85
```

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

8.59.1.14 SD_SPI_PORT

```
#define SD_SPI_PORT spi1
```

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

8.59.1.15 SD_MISO_PIN

```
#define SD_MISO_PIN 12
```

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

8.59.1.16 SD_MOSI_PIN

```
#define SD_MOSI_PIN 11
```

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

8.59.1.17 SD_SCK_PIN

```
#define SD_SCK_PIN 10
```

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

8.59.1.18 SD_CS_PIN

```
#define SD_CS_PIN 13
```

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

8.59.1.19 SD_CARD_DETECT_PIN

```
#define SD_CARD_DETECT_PIN 28
```

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

8.59.1.20 SX1278_MISO

```
#define SX1278_MISO 16
```

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

8.59.1.21 SX1278_CS

```
#define SX1278_CS 17
```

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

8.59.1.22 SX1278_SCK

```
#define SX1278_SCK 18
```

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

8.59.1.23 SX1278_MOSI

```
#define SX1278_MOSI 19
```

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

8.59.1.24 SPI_PORT

```
#define SPI_PORT spi0
```

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

8.59.1.25 READ_BIT

```
#define READ_BIT 0x80
```

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

8.59.1.26 LORA_DEFAULT_SPI

```
#define LORA_DEFAULT_SPI spi0
```

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

8.59.1.27 LORA_DEFAULT_SPI_FREQUENCY

```
#define LORA_DEFAULT_SPI_FREQUENCY 8E6
```

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

8.59.1.28 LORA_DEFAULT_SS_PIN

```
#define LORA_DEFAULT_SS_PIN 17
```

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

8.59.1.29 LORA_DEFAULT_RESET_PIN

```
#define LORA_DEFAULT_RESET_PIN 22
```

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

8.59.1.30 LORA_DEFAULT_DIO0_PIN

```
#define LORA_DEFAULT_DIO0_PIN 20
```

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

8.59.1.31 PA_OUTPUT_RFO_PIN

```
#define PA_OUTPUT_RFO_PIN 11
```

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

8.59.1.32 PA_OUTPUT_PA_BOOST_PIN

```
#define PA_OUTPUT_PA_BOOST_PIN 12
```

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

8.59.2 Variable Documentation

8.59.2.1 csPin

```
const int csPin [extern]
```

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

8.59.2.2 resetPin

```
const int resetPin [extern]
```

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

8.59.2.3 irqPin

```
const int irqPin [extern]
```

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

8.59.2.4 localAddress

```
uint8_t localAddress [extern]
```

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

8.59.2.5 destination

```
uint8_t destination [extern]
```

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

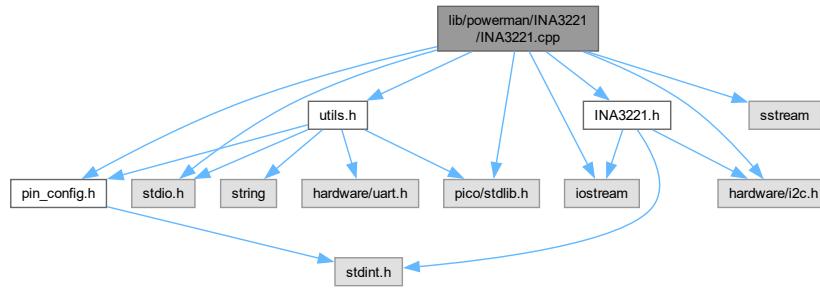
8.60 pin_config.h

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

```
00001 // pin_config.h
00002 #include <stdint.h>
00003
00004 #ifndef PIN_CONFIG_H
00005 #define PIN_CONFIG_H
00006
00007 //DEBUG uart
00008 #define DEBUG_UART_PORT uart0
00009 #define DEBUG_UART_BAUD_RATE 115200
00010
00011 #define DEBUG_UART_TX_PIN 0
00012 #define DEBUG_UART_RX_PIN 1
00013
00014 #define MAIN_I2C_PORT i2c1
00015 #define MAIN_I2C_SDA_PIN 6
00016 #define MAIN_I2C_SCL_PIN 7
00017
00018 // GPS configuration
00019 #define GPS_UART_PORT uart1
00020 #define GPS_UART_BAUD_RATE 9600
00021 #define GPS_UART_TX_PIN 8
00022 #define GPS_UART_RX_PIN 9
00023 #define GPS_POWER_ENABLE_PIN 14
00024
00025 #define BUFFER_SIZE 85 // NMEA sentences are usually under 85 chars
00026
00027 // SPI configuration for SD card
00028 #define SD_SPI_PORT spi1
00029 #define SD_MISO_PIN 12
00030 #define SD_MOSI_PIN 11
00031 #define SD_SCK_PIN 10
00032 #define SD_CS_PIN 13
00033 #define SD_CARD_DETECT_PIN 28
00034
00035 #define SX1278_MISO 16
00036 #define SX1278_CS 17
00037 #define SX1278_SCK 18
00038 #define SX1278_MOSI 19
00039
00040 #define SPI_PORT spi0
00041 #define READ_BIT 0x80
00042
00043 #define LORA_DEFAULT_SPI spi0
00044 #define LORA_DEFAULT_SPI_FREQUENCY 8E6
00045 #define LORA_DEFAULT_SS_PIN 17
00046 #define LORA_DEFAULT_RESET_PIN 22
00047 #define LORA_DEFAULT_DIO0_PIN 20
00048
00049 #define PA_OUTPUT_RFO_PIN 11
00050 #define PA_OUTPUT_PA_BOOST_PIN 12
00051
00052
00053
00054 // LoRa constants - declare as extern
00055 extern const int cspin; // LoRa radio chip select
00056 extern const int resetPin; // LoRa radio reset
00057 extern const int irqPin; // LoRa hardware interrupt pin
00058 extern uint8_t localAddress; // address of this device
00059 extern uint8_t destination; // destination to send to
00060
00061
00062 #endif // PIN_CONFIG_H
```

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

```
#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:
```



8.62 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 INA3221::INA3221(ina3221_addr_t addr, i2c_inst_t* i2c)
00011     : _i2c_addr(addr), _i2c(i2c) {}
00012
00013
00014 bool INA3221::begin() {
00015     uartPrint("INA3221 initializing...\n");
00016
00017     _shuntRes[0] = 10;
00018     _shuntRes[1] = 10;
00019     _shuntRes[2] = 10;
00020
00021     _filterRes[0] = 10;
00022     _filterRes[1] = 10;
00023     _filterRes[2] = 10;
00024
00025     uint16_t manuf_id = getManufID();
00026     uint16_t die_id = getDieID();
00027     std::stringstream ss;
00028     ss << "INA3221 Manufacturer ID: 0x" << std::hex << manuf_id
00029         << ", Die ID: 0x" << die_id << std::endl;
00030     uartPrint(ss.str());
00031
00032     if (manuf_id == 0x5449 && die_id == 0x3220) {
00033         uartPrint("INA3221 found and initialized.");
00034         return true;
00035     } else {
00036         uartPrint("INA3221 initialization failed. Incorrect IDs.");
00037         return false;
00038     }
00039 }
```

```
00039
00040 }
00041
00042 void INA3221::_read(inA3221_reg_t reg, uint16_t *val) {
00043     uint8_t reg_buf = reg;
00044     uint8_t data[2];
00045
00046     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, &reg_buf, 1, true);
00047     if (ret != 1) {
00048         std::cerr << "Failed to write register address to I2C device." << std::endl;
00049         return;
00050     }
00051
00052     ret = i2c_read_blocking(MAIN_I2C_PORT, _i2c_addr, data, 2, false);
00053     if (ret != 2) {
00054         std::cerr << "Failed to read data from I2C device." << std::endl;
00055         return;
00056     }
00057
00058     *val = (data[0] << 8) | data[1];
00059 }
00060
00061 void INA3221::_write(inA3221_reg_t reg, uint16_t *val) {
00062     uint8_t buf[3];
00063     buf[0] = reg;
00064     buf[1] = (*val >> 8) & 0xFF; // MSB
00065     buf[2] = (*val) & 0xFF; // LSB
00066
00067     int ret = i2c_write_blocking(MAIN_I2C_PORT, _i2c_addr, buf, 3, false);
00068     if (ret != 3) {
00069         std::cerr << "Failed to write data to I2C device." << std::endl;
00070     }
00071 }
00072
00073 uint16_t INA3221::getReg(inA3221_reg_t reg){
00074     uint16_t val = 0;
00075     _read(reg, &val);
00076     return val;
00077 }
00078
00079 void INA3221::reset(){
00080     conf_reg_t conf_reg;
00081
00082     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00083     conf_reg.reset = 1;
00084     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00085 }
00086
00087 void INA3221::setModePowerDown(){
00088     conf_reg_t conf_reg;
00089
00090     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00091     conf_reg.mode_bus_en = 0;
00092     conf_reg.mode_continious_en = 0 ;
00093     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00094 }
00095
00096 void INA3221::setModeContinious(){
00097     conf_reg_t conf_reg;
00098
00099     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00100     conf_reg.mode_continious_en = 1;
00101     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00102 }
00103
00104 void INA3221::setModeTriggered(){
00105     conf_reg_t conf_reg;
00106
00107     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00108     conf_reg.mode_continious_en = 0;
00109     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00110 }
00111
00112 void INA3221::setShuntMeasEnable(){
00113     conf_reg_t conf_reg;
00114
00115     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00116     conf_reg.mode_shunt_en = 1;
00117     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00118 }
00119
00120 void INA3221::setShuntMeasDisable(){
00121     conf_reg_t conf_reg;
00122
00123     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00124     conf_reg.mode_shunt_en = 0;
00125     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
```

```

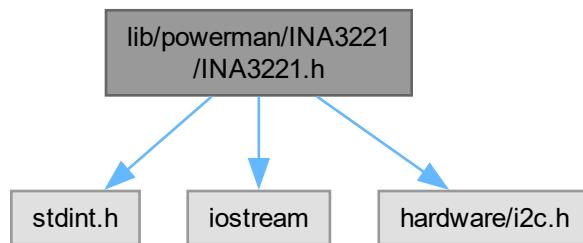
00126 }
00127
00128 void INA3221::setBusMeasEnable() {
00129     conf_reg_t conf_reg;
00130
00131     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00132     conf_reg.mode_bus_en = 1;
00133     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00134 }
00135
00136 void INA3221::setBusMeasDisable() {
00137     conf_reg_t conf_reg;
00138
00139     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00140     conf_reg.mode_bus_en = 0;
00141     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00142 }
00143
00144 void INA3221::setAveragingMode(ina3221_avg_mode_t mode) {
00145     conf_reg_t conf_reg;
00146
00147     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00148     conf_reg.avg_mode = mode;
00149     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00150 }
00151
00152 void INA3221::setBusConversionTime(ina3221_conv_time_t convTime) {
00153     conf_reg_t conf_reg;
00154
00155     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00156     conf_reg.bus_conv_time = convTime;
00157     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00158 }
00159
00160 void INA3221::setShuntConversionTime(ina3221_conv_time_t convTime) {
00161     conf_reg_t conf_reg;
00162
00163     _read(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00164     conf_reg.shunt_conv_time = convTime;
00165     _write(INA3221_REG_CONF, (uint16_t*)&conf_reg);
00166 }
00167
00168 uint16_t INA3221::getManufID() {
00169     uint16_t id = 0;
00170     _read(INA3221_REG_MANUF_ID, &id);
00171     return id;
00172 }
00173
00174 uint16_t INA3221::getDieID() {
00175     uint16_t id = 0;
00176     _read(INA3221_REG_DIE_ID, &id);
00177     return id;
00178 }
00179
00180 int32_t INA3221::getShuntVoltage(ina3221_ch_t channel) {
00181     int32_t res;
00182     ina3221_reg_t reg;
00183     uint16_t val_raw = 0;
00184
00185     switch(channel) {
00186         case INA3221_CH1:
00187             reg = INA3221_REG_CH1_SHUNTV;
00188             break;
00189         case INA3221_CH2:
00190             reg = INA3221_REG_CH2_SHUNTV;
00191             break;
00192         case INA3221_CH3:
00193             reg = INA3221_REG_CH3_SHUNTV;
00194             break;
00195     }
00196
00197     _read(reg, &val_raw);
00198
00199     res = (int16_t) (val_raw >> 3);
00200     res *= SHUNT_VOLTAGE_LSB_UV;
00201
00202     return res;
00203 }
00204
00205 int32_t INA3221::estimateOffsetVoltage(ina3221_ch_t channel, uint32_t busV) {
00206     float bias_in = 10.0;           // Input bias current at IN- in uA
00207     float r_in = 0.670;            // Input resistance at IN- in MOhm
00208     uint32_t adc_step = 40;        // smallest shunt ADC step in uV
00209     float shunt_res = _shuntRes[channel]/1000.0; // convert to Ohm
00210     float filter_res = _filterRes[channel];
00211     int32_t offset = 0.0;
00212     float reminder;

```

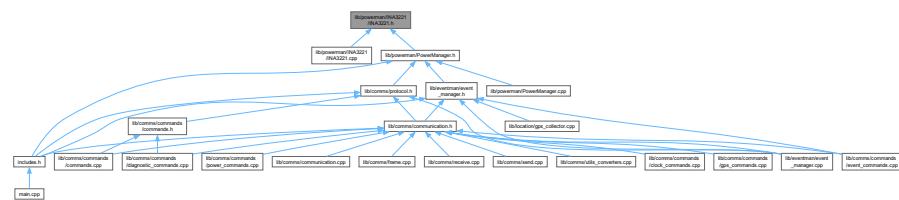
```
00213
00214     offset = (shunt_res + filter_res)*(busV/r_in + bias_in) - bias_in * filter_res;
00215
00216     // Round the offset to the closest shunt ADC value
00217     remainder = offset % adc_step;
00218     if (remainder < adc_step/2)
00219     {
00220         offset -= remainder;
00221     } else {
00222         offset += adc_step - remainder;
00223     }
00224
00225     return offset;
00226 }
00227
00228 float INA3221::getCurrent(ina3221_ch_t channel) {
00229     int32_t shunt_uV = 0;
00230     float current_A = 0;
00231
00232     shunt_uV = getShuntVoltage(channel);
00233     current_A = shunt_uV / (int32_t)_shuntRes[channel] / 1000.0;
00234     return current_A;
00235 }
00236
00237 float INA3221::getCurrent_mA(ina3221_ch_t channel) {
00238     int32_t shunt_uV = 0;
00239     float current_A = 0;
00240
00241     shunt_uV = getShuntVoltage(channel);
00242     current_A = shunt_uV / (int32_t)_shuntRes[channel];
00243     return current_A;
00244 }
00245
00246 float INA3221::getCurrentCompensated(ina3221_ch_t channel) {
00247     int32_t shunt_uV = 0;
00248     int32_t bus_V = 0;
00249     float current_A = 0.0;
00250     int32_t offset_uV = 0;
00251
00252     shunt_uV = getShuntVoltage(channel);
00253     bus_V = getVoltage(channel);
00254     offset_uV = estimateOffsetVoltage(channel, bus_V);
00255
00256     current_A = (shunt_uV - offset_uV) / (int32_t)_shuntRes[channel] / 1000.0;
00257
00258     return current_A;
00259 }
00260
00261 float INA3221::getCurrentCompensated_mA(ina3221_ch_t channel) {
00262     int32_t shunt_uV = 0;
00263     int32_t bus_V = 0;
00264     float current_A = 0.0;
00265     int32_t offset_uV = 0;
00266
00267     shunt_uV = getShuntVoltage(channel);
00268     bus_V = getVoltage(channel);
00269     offset_uV = estimateOffsetVoltage(channel, bus_V);
00270
00271     current_A = (shunt_uV - offset_uV) / (int32_t)_shuntRes[channel];
00272
00273     return current_A;
00274 }
00275
00276 float INA3221::getVoltage(ina3221_ch_t channel) {
00277     float voltage_V = 0.0;
00278     ina3221_reg_t reg;
00279     uint16_t val_raw = 0;
00280
00281     switch(channel){
00282         case INA3221_CH1:
00283             reg = INA3221_REG_CH1_BUSV;
00284             break;
00285         case INA3221_CH2:
00286             reg = INA3221_REG_CH2_BUSV;
00287             break;
00288         case INA3221_CH3:
00289             reg = INA3221_REG_CH3_BUSV;
00290             break;
00291     }
00292
00293     _read(reg, &val_raw);
00294     voltage_V = val_raw / 1000.0;
00295     return voltage_V;
00296 }
```

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

```
#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`
 - struct `INA3221::conf_reg_t`
 - struct `INA3221::masken_reg_t`

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_CH3_WARNING_ALERT_LIM , INA3221_REG_SHUNTV_SUM , INA3221_REG_SHUNTV_SUM_LIM
,
INA3221_REG_MASK_ENABLE ,
INA3221_REG_PWR_VALID_HI_LIM , INA3221_REG_PWR_VALID_LO_LIM , INA3221_REG_MANUF_ID
= 0xFE , INA3221_REG_DIE_ID = 0xFF }
```

- 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` }
- 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` }

**Variables**

- `const int INA3221_CH_NUM` = 3
- `const int SHUNT_VOLTAGE_LSB_UV` = 5

**8.63.1 Enumeration Type Documentation****8.63.1.1 ina3221\_addr\_t**

enum `ina3221_addr_t`

**Enumerator**

|                                 |  |
|---------------------------------|--|
| <code>INA3221_ADDR40_GND</code> |  |
| <code>INA3221_ADDR41_VCC</code> |  |
| <code>INA3221_ADDR42_SDA</code> |  |
| <code>INA3221_ADDR43_SCL</code> |  |

Definition at line 8 of file `INA3221.h`.

**8.63.1.2 ina3221\_ch\_t**

enum `ina3221_ch_t`

**Enumerator**

|                          |  |
|--------------------------|--|
| <code>INA3221_CH1</code> |  |
| <code>INA3221_CH2</code> |  |
| <code>INA3221_CH3</code> |  |

Definition at line 16 of file `INA3221.h`.

**8.63.1.3 ina3221\_reg\_t**

enum `ina3221_reg_t`

## 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 26 of file [INA3221.h](#).

#### 8.63.1.4 ina3221\_conv\_time\_t

```
enum ina3221_conv_time_t
```

## 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 50 of file [INA3221.h](#).

#### 8.63.1.5 ina3221\_avg\_mode\_t

```
enum ina3221_avg_mode_t
```

## Enumerator

|                        |
|------------------------|
| INA3221_REG_CONF_AVG_1 |
|                        |

## Enumerator

|                           |  |
|---------------------------|--|
| 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 62 of file [INA3221.h](#).

## 8.63.2 Variable Documentation

### 8.63.2.1 INA3221\_CH\_NUM

```
const int INA3221_CH_NUM = 3
```

Definition at line 22 of file [INA3221.h](#).

### 8.63.2.2 SHUNT\_VOLTAGE\_LSB\_UV

```
const int SHUNT_VOLTAGE_LSB_UV = 5
```

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

## 8.64 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
00008 typedef enum {
00009 INA3221_ADDR40_GND = 0b1000000, // A0 pin -> GND
00010 INA3221_ADDR41_VCC = 0b1000001, // A0 pin -> VCC
00011 INA3221_ADDR42_SDA = 0b1000010, // A0 pin -> SDA
00012 INA3221_ADDR43_SCL = 0b1000011 // A0 pin -> SCL
00013 } ina3221_addr_t;
00014
00015 // Channels
00016 typedef enum {
00017 INA3221_CH1 = 0,
00018 INA3221_CH2,
00019 INA3221_CH3,
00020 } ina3221_ch_t;
00021
00022 const int INA3221_CH_NUM = 3;
00023 const int SHUNT_VOLTAGE_LSB_UV = 5;
00024
00025 // Registers
00026 typedef enum {
00027 INA3221_REG_CONF = 0,
00028 INA3221_REG_CH1_SHUNTV,
00029 INA3221_REG_CH1_BUSV,
00030 INA3221_REG_CH2_SHUNTV,
```

```

00031 INA3221_REG_CH2_BUSV,
00032 INA3221_REG_CH3_SHUNTV,
00033 INA3221_REG_CH3_BUSV,
00034 INA3221_REG_CH1_CRIT_ALERT_LIM,
00035 INA3221_REG_CH1_WARNING_ALERT_LIM,
00036 INA3221_REG_CH2_CRIT_ALERT_LIM,
00037 INA3221_REG_CH2_WARNING_ALERT_LIM,
00038 INA3221_REG_CH3_CRIT_ALERT_LIM,
00039 INA3221_REG_CH3_WARNING_ALERT_LIM,
00040 INA3221_REG_SHUNTV_SUM,
00041 INA3221_REG_SHUNTV_SUM_LIM,
00042 INA3221_REG_MASK_ENABLE,
00043 INA3221_REG_PWR_VALID_HI_LIM,
00044 INA3221_REG_PWR_VALID_LO_LIM,
00045 INA3221_REG_MANUF_ID = 0xFF,
00046 INA3221_REG_DIE_ID = 0xFF
00047 } ina3221_reg_t;
00048
00049 // Conversion times
00050 typedef enum {
00051 INA3221_REG_CONF_CT_140US = 0,
00052 INA3221_REG_CONF_CT_204US,
00053 INA3221_REG_CONF_CT_332US,
00054 INA3221_REG_CONF_CT_588US,
00055 INA3221_REG_CONF_CT_1100US,
00056 INA3221_REG_CONF_CT_2116US,
00057 INA3221_REG_CONF_CT_4156US,
00058 INA3221_REG_CONF_CT_8244US
00059 } ina3221_conv_time_t;
00060
00061 // Averaging modes
00062 typedef enum {
00063 INA3221_REG_CONF_AVG_1 = 0,
00064 INA3221_REG_CONF_AVG_4,
00065 INA3221_REG_CONF_AVG_16,
00066 INA3221_REG_CONF_AVG_64,
00067 INA3221_REG_CONF_AVG_128,
00068 INA3221_REG_CONF_AVG_256,
00069 INA3221_REG_CONF_AVG_512,
00070 INA3221_REG_CONF_AVG_1024
00071 } ina3221_avg_mode_t;
00072
00073 class INA3221 {
00074
00075 // Configuration register
00076 typedef struct {
00077 uint16_t mode_shunt_en:1;
00078 uint16_t mode_bus_en:1;
00079 uint16_t mode_continuous_en:1;
00080 uint16_t shunt_conv_time:3;
00081 uint16_t bus_conv_time:3;
00082 uint16_t avg_mode:3;
00083 uint16_t ch3_en:1;
00084 uint16_t ch2_en:1;
00085 uint16_t ch1_en:1;
00086 uint16_t reset:1;
00087 } conf_reg_t;
00088
00089 // Mask/Enable register
00090 typedef struct {
00091 uint16_t conv_ready:1;
00092 uint16_t timing_ctrl_alert:1;
00093 uint16_t pwr_valid_alert:1;
00094 uint16_t warn_alert_ch3:1;
00095 uint16_t warn_alert_ch2:1;
00096 uint16_t warn_alert_ch1:1;
00097 uint16_t shunt_sum_alert:1;
00098 uint16_t crit_alert_ch3:1;
00099 uint16_t crit_alert_ch2:1;
00100 uint16_t crit_alert_ch1:1;
00101 uint16_t crit_alert_latch_en:1;
00102 uint16_t warn_alert_latch_en:1;
00103 uint16_t shunt_sum_en_ch3:1;
00104 uint16_t shunt_sum_en_ch2:1;
00105 uint16_t shunt_sum_en_ch1:1;
00106 uint16_t reserved:1;
00107 } masken_reg_t;
00108
00109 i2c_inst_t* _i2c;
00110 // I2C address
00111 ina3221_addr_t _i2c_addr;
00112
00113 // Shunt resistance in mOhm
00114 uint32_t _shuntRes[INA3221_CH_NUM];
00115
00116 // Series filter resistance in Ohm
00117 uint32_t _filterRes[INA3221_CH_NUM];

```

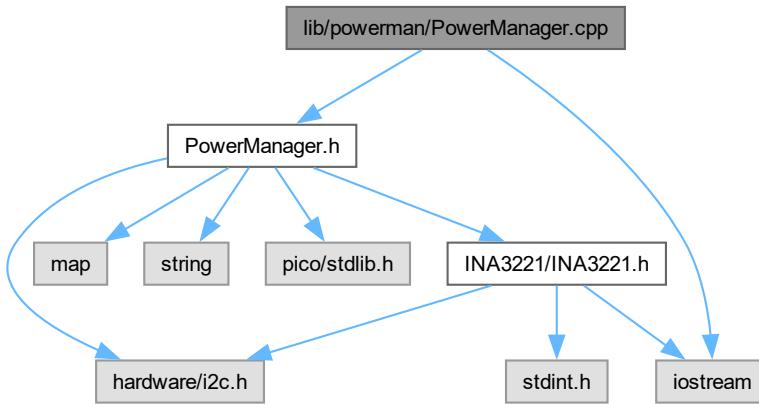
```
00118 // Value of Mask/Enable register.
00119 masken_reg_t _masken_reg;
00120
00121 // Reads 16 bytes from a register.
00122 void _read(in3221_reg_t reg, uint16_t *val);
00123
00124 // Writes 16 bytes to a register.
00125 void _write(in3221_reg_t reg, uint16_t *val);
00126
00127 public:
00128 INA3221(in3221_addr_t addr, i2c_inst_t* i2c);
00129 // Initializes INA3221
00130 bool begin();
00131
00132 // Sets shunt resistor value in mOhm
00133 void setShuntRes(uint32_t res_ch1, uint32_t res_ch2, uint32_t res_ch3);
00134
00135 // Sets filter resistors value in Ohm
00136 void setFilterRes(uint32_t res_ch1, uint32_t res_ch2, uint32_t res_ch3);
00137
00138 // Sets I2C address of INA3221
00139 void setAddr(in3221_addr_t addr) { _i2c_addr = addr; }
00140
00141 // Gets a register value.
00142 uint16_t getReg(in3221_reg_t reg);
00143
00144 // Resets INA3221
00145 void reset();
00146
00147 // Sets operating mode to power-down
00148 void setModePowerDown();
00149
00150 // Sets operating mode to continious
00151 void setModeContinious();
00152
00153 // Sets operating mode to triggered (single-shot)
00154 void setModeTriggered();
00155
00156 // Enables shunt-voltage measurement
00157 void setShuntMeasEnable();
00158
00159 // Disables shunt-voltage mesurement
00160 void setShuntMeasDisable();
00161
00162 // Enables bus-voltage measurement
00163 void setBusMeasEnable();
00164
00165 // Disables bus-voltage measureement
00166 void setBusMeasDisable();
00167
00168 // Sets averaging mode. Sets number of samples that are collected
00169 // and averaged togehter.
00170 void setAveragingMode(in3221_avg_mode_t mode);
00171
00172 // Sets bus-voltage conversion time.
00173 void setBusConversionTime(in3221_conv_time_t convTime);
00174
00175 // Sets shunt-voltage conversion time.
00176 void setShuntConversionTime(in3221_conv_time_t convTime);
00177
00178 // Sets critical alert limit for a channel
00179 void setCritAlertLimit(in3221_ch_t channel, int32_t voltageuV);
00180
00181 // Sets warning alert limit for a channel
00182 void setWarnAlertLimit(in3221_ch_t channel, int32_t voltageuV);
00183
00184 // Sets power-valid upper-limit voltage. The power-valid condition
00185 // is reached when all bus-voltage channels exceed the value set.
00186 // When the powervalid condition is met, the PV alert pin asserts high.
00187 void setPwrValidUpLimit(int16_t voltagemV);
00188
00189 // Sets power-valid lower-limit voltage. If any bus-voltage channel drops
00190 // below the power-valid lower-limit, the PV alert pin pulls low.
00191 void setPwrValidLowLimit(int16_t voltagemV);
00192
00193 // Sets the value that is compared to the Shunt-Voltage Sum register value
00194 // following each completed cycle of all selected channels to detect
00195 // for system overcurrent events.
00196 void setShuntSumAlertLimit(int32_t voltagemV);
00197
00198 // Sets the current value that is compared to the sum all currents.
00199 // This function is a helper for setShuntSumAlertLim(). It onverts current
00200 // value to shunt voltage value.
00201 void setCurrentSumAlertLimit(int32_t currentmA);
```

```
00205 // Enables warning alert latch.
00206 void setWarnAlertLatchEnable();
00207
00208 // Disables warning alert latch.
00209 void setWarnAlertLatchDisable();
00210
00211 // Enables critical alert latch.
00212 void setCritAlertLatchEnable();
00213
00214 // Disables critical alert latch.
00215 void setCritAlertLatchDisable();
00216
00217 // Reads flags from Mask/Enable register.
00218 // When Mask/Enable register is read, flags are cleared.
00219 // Use getTimingCtrlAlertFlag(), getPwrValidAlertFlag(),
00220 // getCurrentSumAlertFlag() and getConvReadyFlag() to get flags after
00221 // readFlags() is called.
00222 void readFlags();
00223
00224 // Gets timing-control-alert flag indicator.
00225 bool getTimingCtrlAlertFlag();
00226
00227 // Gets power-valid-alert flag indicator.
00228 bool getPwrValidAlertFlag();
00229
00230 // Gets summation-alert flag indicator.
00231 bool getCurrentSumAlertFlag();
00232
00233 // Gets Conversion-ready flag.
00234 bool getConversionReadyFlag();
00235
00236 // Gets manufacturer ID.
00237 // Should read 0x5449.
00238 uint16_t getManufID();
00239
00240 // Gets die ID.
00241 // Should read 0x3220.
00242 uint16_t getDieID();
00243
00244 // Enables channel measurements
00245 void setChannelEnable(ina3221_ch_t channel);
00246
00247 // Disables channel measurements
00248 void setChannelDisable(ina3221_ch_t channel);
00249
00250 // Sets warning alert shunt voltage limit
00251 void setWarnAlertShuntLimit(ina3221_ch_t channel, int32_t voltageuV);
00252
00253 // Sets critical alert shunt voltage limit
00254 void setCritAlertShuntLimit(ina3221_ch_t channel, int32_t voltageuV);
00255
00256 // Sets warning alert current limit
00257 void setWarnAlertCurrentLimit(ina3221_ch_t channel, int32_t currentmA);
00258
00259 // Sets critical alert current limit
00260 void setCritAlertCurrentLimit(ina3221_ch_t channel, int32_t currentmA);
00261
00262 // Includes channel to fill Shunt-Voltage Sum register.
00263 void setCurrentSumEnable(ina3221_ch_t channel);
00264
00265 // Excludes channel from filling Shunt-Voltage Sum register.
00266 void setCurrentSumDisable(ina3221_ch_t channel);
00267
00268 // Gets shunt voltage in uV.
00269 int32_t getShuntVoltage(ina3221_ch_t channel);
00270
00271 // Gets warning alert flag.
00272 bool getWarnAlertFlag(ina3221_ch_t channel);
00273
00274 // Gets critical alert flag.
00275 bool getCritAlertFlag(ina3221_ch_t channel);
00276
00277 // Estimates offset voltage added by the series filter resistors
00278 int32_t estimateOffsetVoltage(ina3221_ch_t channel, uint32_t busVoltage);
00279
00280 // Gets current in A.
00281 float getCurrent(ina3221_ch_t channel);
00282
00283 float getCurrent_mA(ina3221_ch_t channel);
00284
00285 // Gets current compensated with calculated offset voltage.
00286 float getCurrentCompensated(ina3221_ch_t channel);
00287
00288 float getCurrentCompensated_mA(ina3221_ch_t channel);
00289 // Gets bus voltage in V.
00290 float getVoltage(ina3221_ch_t channel);
00291 };
```

```
00292
00293 #endif
```

## 8.65 lib/powerman/PowerManager.cpp File Reference

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



## 8.66 PowerManager.cpp

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

```
00001 #include "PowerManager.h"
00002 #include <iostream>
00003
00004 PowerManager::PowerManager(i2c_inst_t* i2c)
00005 : ina3221(INA3221_ADDR40_GND, i2c) {}
00006
00007 bool PowerManager::initialize() {
00008 initialized = ina3221.begin();
00009 return initialized;
00010 }
00011
00012 std::string PowerManager::readIDs() {
00013 if (!initialized) return "noinit";
00014 std::string MAN = "MAN " + std::to_string(ina3221.getManufID());
00015 std::string DIE = "DIE " + std::to_string(ina3221.getDieID());
00016 return MAN + " - " + DIE;
00017 }
00018
00019 float PowerManager::getVoltageBattery() {
00020 if (!initialized) return 0.0f;
00021 return ina3221.getVoltage(INA3221_CH1);
00022 }
00023
00024 float PowerManager::getVoltage5V() {
00025 if (!initialized) return 0.0f;
00026 return ina3221.getVoltage(INA3221_CH2);
00027 }
00028
00029 float PowerManager::getCurrentChargeUSB() {
00030 if (!initialized) return 0.0f;
00031 return ina3221.getCurrent_mA(INA3221_CH1);
00032 }
```

```

00033
00034 float PowerManager::getCurrentDraw() {
00035 if (!initialized) return 0.0f;
00036 return ina3221.getCurrent_mA(INA3221_CH2);
00037 }
00038
00039 float PowerManager::getCurrentChargeSolar() {
00040 if (!initialized) return 0.0f;
00041 return ina3221.getCurrent_mA(INA3221_CH3);
00042 }
00043
00044 float PowerManager::getCurrentChargeTotal() {
00045 if (!initialized) return 0.0f;
00046 return ina3221.getCurrent_mA(INA3221_CH1) + ina3221.getCurrent_mA(INA3221_CH3);
00047 }
00048
00049 void PowerManager::configure(const std::map<std::string, std::string>& config) {
00050 if (!initialized) return;
00051
00052 if (config.find("operating_mode") != config.end()) {
00053 if (config.at("operating_mode") == "continuous") {
00054 ina3221.setModeContinious();
00055 }
00056 }
00057
00058 if (config.find("averaging_mode") != config.end()) {
00059 int avg_mode = std::stoi(config.at("averaging_mode"));
00060 switch(avg_mode) {
00061 case 1:
00062 ina3221.setAveragingMode(INA3221_REG_CONF_AVG_1);
00063 break;
00064 case 4:
00065 ina3221.setAveragingMode(INA3221_REG_CONF_AVG_4);
00066 break;
00067 case 16:
00068 ina3221.setAveragingMode(INA3221_REG_CONF_AVG_16);
00069 break;
00070 default:
00071 ina3221.setAveragingMode(INA3221_REG_CONF_AVG_16);
00072 }
00073 }
00074 }
00075
00076 bool PowerManager::isSolarActive() {
00077 if (!initialized) return false;
00078 return getCurrentChargeSolar() > SOLAR_CURRENT_THRESHOLD;
00079 }
00080
00081 bool PowerManager::isUSBConnected() {
00082 if (!initialized) return false;
00083 return getCurrentChargeUSB() > USB_CURRENT_THRESHOLD;
00084 }

```

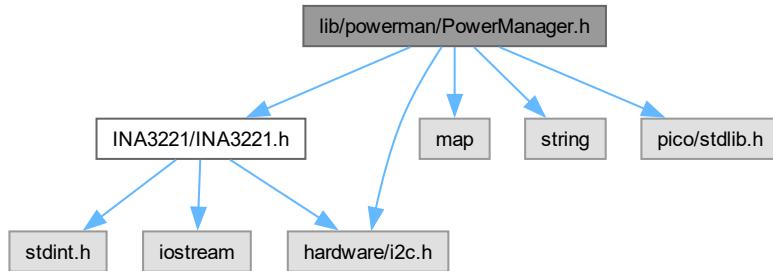
## 8.67 lib/powerman/PowerManager.h File Reference

```

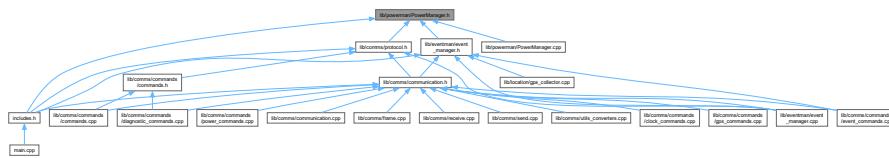
#include "INA3221/INA3221.h"
#include <map>
#include <string>
#include <hardware/i2c.h>
#include "pico/stdlib.h"

```

Include dependency graph for PowerManager.h:



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



## Classes

- class [PowerManager](#)

## 8.68 PowerManager.h

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

```

00001 // [lib/PowerManager/PowerManager.h] (lib/PowerManager/PowerManager.h)
00002 #ifndef POWER_MANAGER_H
00003 #define POWER_MANAGER_H
00004
00005 #include "INA3221/INA3221.h"
00006 #include <map>
00007 #include <string>
00008 #include <hardware/i2c.h>
00009 #include "pico/stdcib.h"
00010
0011 class PowerManager {
0012 public:
0013 PowerManager(i2c_inst_t* i2c);
0014 bool initialize();
0015 std::string readIDs();
0016 float getCurrentChargeSolar();
0017 float getCurrentChargeUSB();
0018 float getCurrentChargeTotal();
0019 float getCurrentDraw();
0020 float getVoltageBattery();
0021 float getVoltage5V();
0022 void configure(const std::map<std::string, std::string>& config);
0023 bool isSolarActive();
0024 bool isUSBConnected();
0025
0026 static constexpr float SOLAR_CURRENT_THRESHOLD = 50.0f; // mA
0027 static constexpr float USB_CURRENT_THRESHOLD = 50.0f; // mA
0028 static constexpr float VOLTAGE_LOW_THRESHOLD = 4.7f; // V

```

```

00029 static constexpr float VOLTAGE_OVERCHARGE_THRESHOLD = 5.3f; // V
00030 static constexpr float FALL_RATE_THRESHOLD = -0.02f; // V/sample
00031 static constexpr int FALLING_TREND_REQUIRED = 3; // samples
00032
00033 private:
00034 INA3221 ina3221;
00035 bool initialized;
00036
00037 bool solarActive = false;
00038 bool usbConnected = false;
00039 };
00040
00041 #endif // POWER_MANAGER_H

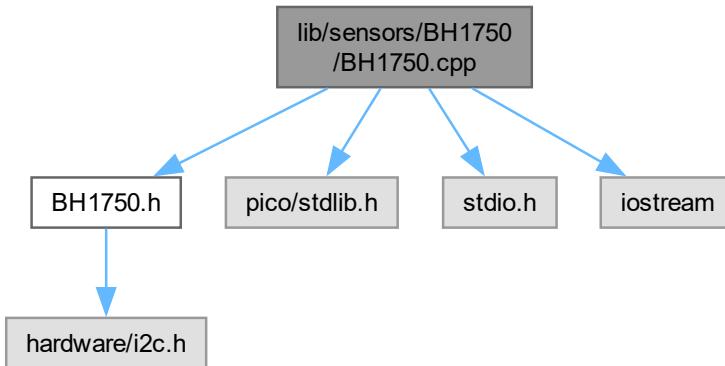
```

## 8.69 lib/sensors/BH1750/BH1750.cpp File Reference

```

#include "BH1750.h"
#include "pico/stdlib.h"
#include <stdio.h>
#include <iostream>
Include dependency graph for BH1750.cpp:

```



## 8.70 BH1750.cpp

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

```

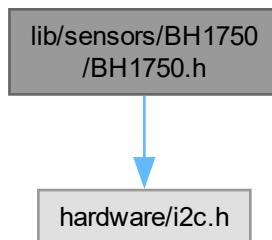
00001 #include "BH1750.h"
00002 #include "pico/stdlib.h"
00003 #include <stdio.h>
00004 #include <iostream>
00005
00006 BH1750::BH1750(uint8_t addr) : _i2c_addr(addr) {}
00007
00008 bool BH1750::begin(Mode mode) {
00009 write8(static_cast<uint8_t>(Mode::POWER_ON));
00010 write8(static_cast<uint8_t>(Mode::RESET));
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 }

```

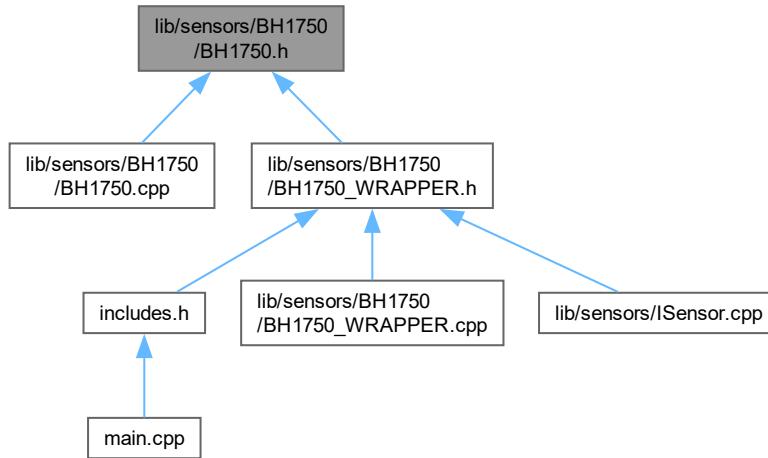
```
00019 return false;
00020 }
00021
00022 void BH1750::configure(Mode mode) {
00023 uint8_t modeVal = static_cast<uint8_t>(mode);
00024 switch (mode) {
00025 case Mode::CONTINUOUS_HIGH_RES_MODE:
00026 case Mode::CONTINUOUS_HIGH_RES_MODE_2:
00027 case Mode::CONTINUOUS_LOW_RES_MODE:
00028 case Mode::ONE_TIME_HIGH_RES_MODE:
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::readLightLevel() {
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 }
```

## 8.71 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`

### 8.71.1 Macro Definition Documentation

#### 8.71.1.1 `_BH1750_DEVICE_ID`

```
#define _BH1750_DEVICE_ID 0xE1
```

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

#### 8.71.1.2 `_BH1750_MTREG_MIN`

```
#define _BH1750_MTREG_MIN 31
```

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

### 8.71.1.3 \_BH1750\_MTREG\_MAX

```
#define _BH1750_MTREG_MAX 254
```

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

### 8.71.1.4 \_BH1750\_DEFAULT\_MTREG

```
#define _BH1750_DEFAULT_MTREG 69
```

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

## 8.72 BH1750.h

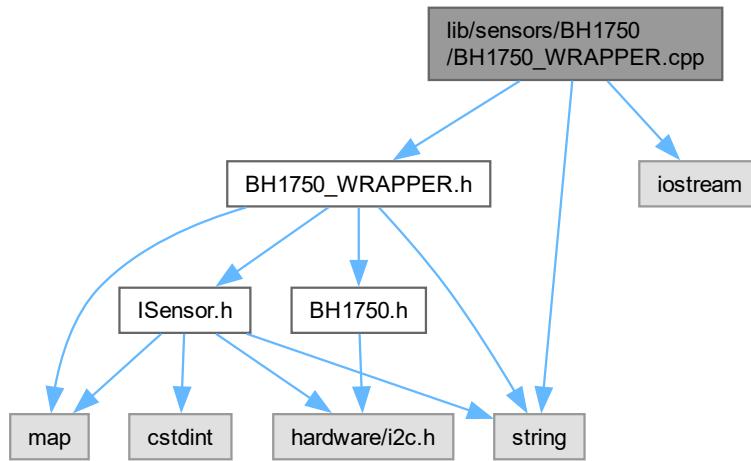
[Go to the documentation of this file.](#)

```
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 readLightLevel();
00031
00032 private:
00033 void write8(uint8_t data);
00034 uint8_t _i2c_addr;
00035 };
00036
00037 #endif // __BH1750_H__
```

## 8.73 lib/sensors/BH1750/BH1750\_WRAPPER.cpp File Reference

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

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



## 8.74 BH1750\_WRAPPER.cpp

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

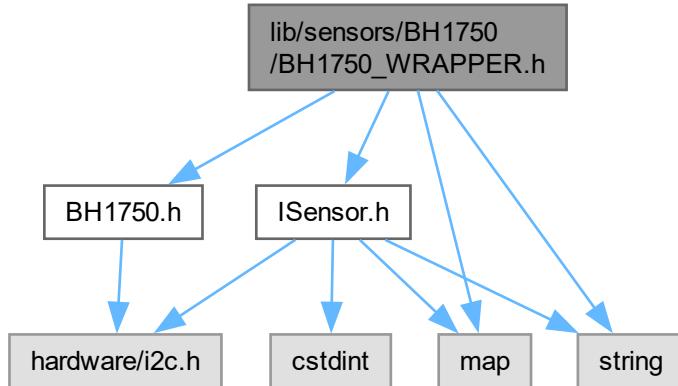
```

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::readData(SensorDataTypeIdentifier type) {
00016 if (type == SensorDataTypeIdentifier::LIGHT_LEVEL) {
00017 return sensor.readLightLevel();
00018 }
00019 return 0.0f;
00020 }
00021
00022 bool BH1750Wrapper::isInitialized() const {
00023 return initialized;
00024 }
00025
00026 SensorType BH1750Wrapper::getType() 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 }
```

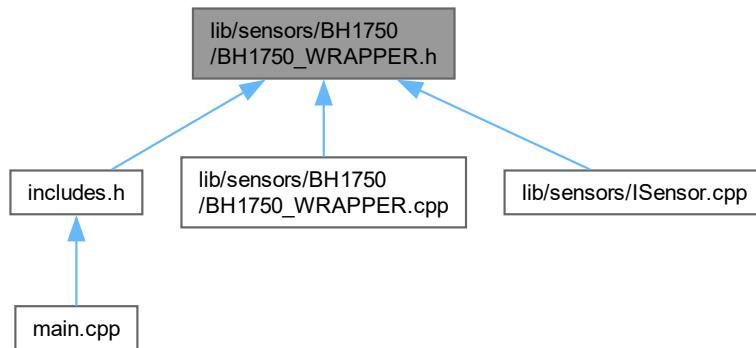
```
00042 else if (value == "one_time_high_resolution") {
00043 sensor.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE);
00044 }
00045 else if (value == "one_time_high_resolution_2") {
00046 sensor.configure(BH1750::Mode::ONE_TIME_HIGH_RES_MODE_2);
00047 }
00048 else if (value == "one_time_low_resolution") {
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 }
```

## 8.75 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](#)

## 8.76 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 readData(SensorDataTypeIdentifier type) override;
00019 bool isInitialized() const override;
00020 SensorType getType() const override;
00021
00022 bool configure(const std::map<std::string, std::string>& config);
00023
00024 };
00025
00026 #endif // BH1750_WRAPPER_H

```

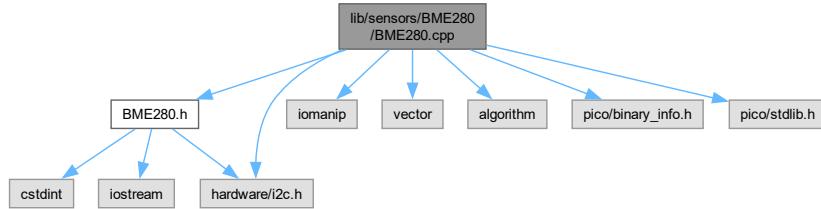
## 8.77 lib/sensors/BME280/BME280.cpp File Reference

```

#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:
```



## 8.78 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, ®, 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, ®, 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, ®, 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

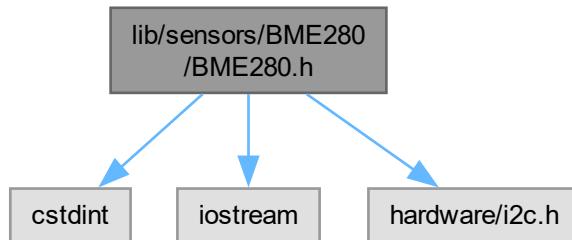
```

```

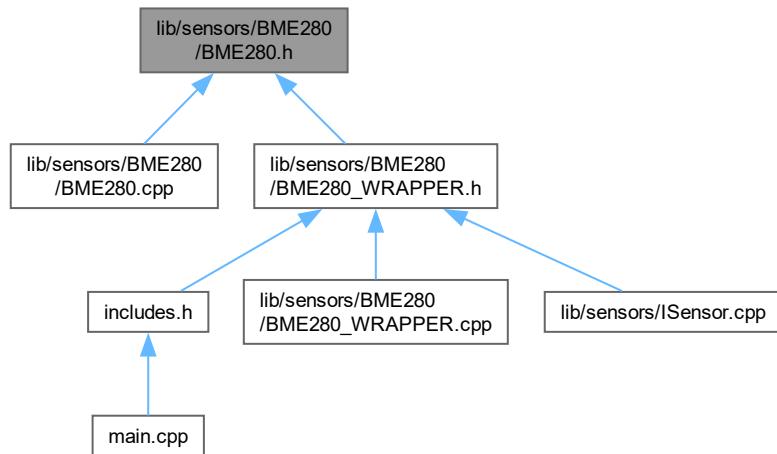
00223 buf[0] = REG_CTRL_MEAS;
00224 buf[1] = 0xB7; // Temp and pressure oversampling x16, normal mode
00225 ret = i2c_write_blocking(i2c_port, device_addr, buf, 2, false);
00226 if (ret !=2) {
00227 std::cerr << "Failed to write CTRL_MEAS to BME280.\n";
00228 return false;
00229 }
00230
00231 return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

## 8.79 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**

## 8.80 BME280.h

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

```

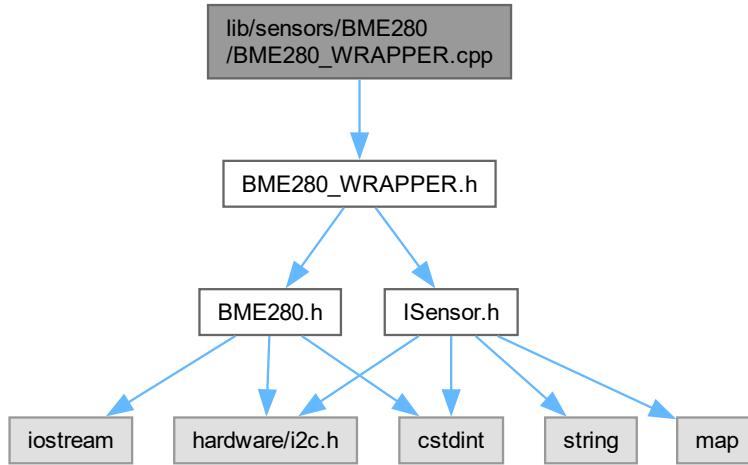
00001 // BME280.h
00002
00003 #ifndef BME280_H
00004 #define BME280_H
00005
00006 #include <cstdint>
00007 #include <iostream>
00008 #include "hardware/i2c.h"
00009
00010 // Calibration parameters structure
00011 struct BME280CalibParam {
00012 // Temperature parameters
00013 uint16_t dig_t1;
00014 int16_t 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
```

## 8.81 lib/sensors/BME280/BME280\_WRAPPER.cpp File Reference

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



## 8.82 BME280\_WRAPPER.cpp

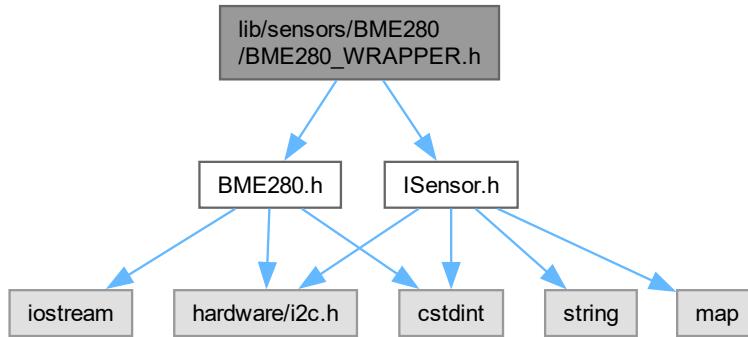
[Go to the documentation of this file.](#)

```

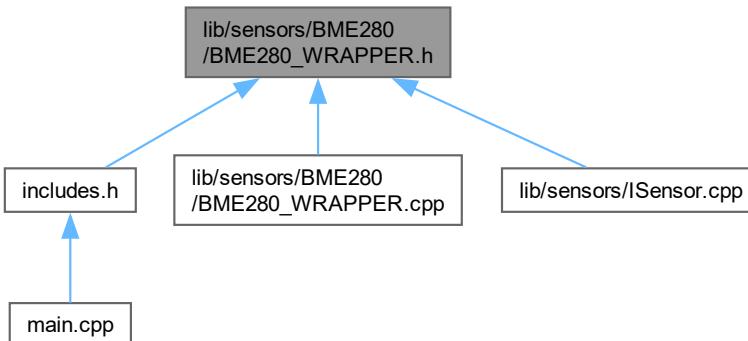
00001 #include "BME280_WRAPPER.h"
00002
00003 BME280Wrapper::BME280Wrapper(i2c_inst_t* i2c) : sensor(i2c) {}
00004
00005 bool BME280Wrapper::init() {
00006 initialized = sensor.init();
00007 return initialized;
00008 }
00009
00010 float BME280Wrapper::readData(SensorDataTypeIdentifier type) {
00011 int32_t 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::isInitialized() const {
00027 return initialized;
00028 }
00029
00030 SensorType BME280Wrapper::getType() const {
00031 return SensorType::ENVIRONMENT;
00032 }
00033
00034 bool BME280Wrapper::configure(const std::map<std::string, std::string>& config) {
00035 return true;
00036 }
```

## 8.83 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](#)

## 8.84 BME280\_WRAPPER.h

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

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

```

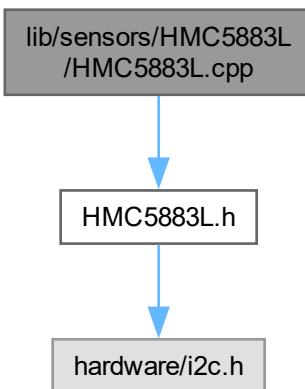
00003 #define BME280_WRAPPER_H
00004
00005 #include "ISensor.h"
00006 #include "BME280.h"
00007
00008 class BME280Wrapper : public ISensor {
00009 private:
0010 BME280 sensor;
0011 bool initialized = false;
0012
0013 public:
0014 BME280Wrapper(i2c_inst_t* i2c);
0015
0016 bool init() override;
0017 float readData(SensorDataTypeIdentifier type) override;
0018 bool isInitialized() const override;
0019 SensorType getType() const override;
0020 bool configure(const std::map<std::string, std::string>& config) override;
0021
0022 };
0023
0024 #endif // BME280_WRAPPER_H

```

## 8.85 lib/sensors/HMC5883L/HMC5883L.cpp File Reference

#include "HMC5883L.h"

Include dependency graph for HMC5883L.cpp:



## 8.86 HMC5883L.cpp

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

```

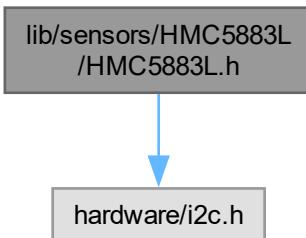
00001 #include "HMC5883L.h"
00002
00003 HMC5883L::HMC5883L(i2c_inst_t* i2c, uint8_t address) : i2c(i2c), address(address) {}
00004
00005 bool HMC5883L::init() {
00006 // Continuous measurement mode, 15Hz data output rate
00007 if (!writeRegister(0x00, 0x70)) return false;
00008 if (!writeRegister(0x01,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

```

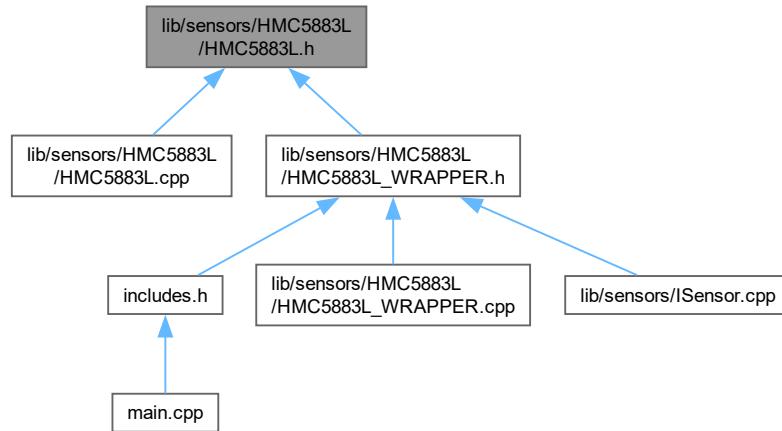
```
00013 bool HMC5883L::read(int16_t& x, int16_t& y, int16_t& z) {
00014 uint8_t buffer[6];
00015 if (!readRegisters(0x03, buffer, 6)) return false;
00016
00017 x = (buffer[0] << 8) | buffer[1];
00018 z = (buffer[2] << 8) | buffer[3];
00019 y = (buffer[4] << 8) | buffer[5];
00020
00021 if (x > 32767) x -= 65536;
00022 if (y > 32767) y -= 65536;
00023 if (z > 32767) z -= 65536;
00024
00025 return true;
00026 }
00027
00028 bool HMC5883L::writeRegister(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::readRegisters(uint8_t reg, uint8_t* buffer, size_t length) {
00034 if (i2c_write_blocking(i2c, address, ®, 1, true) != 1) return false;
00035 return i2c_read_blocking(i2c, address, buffer, length, false) == length;
00036 }
```

## 8.87 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](#)

## 8.88 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 writeRegister(uint8_t reg, uint8_t value);
00017 bool readRegisters(uint8_t reg, uint8_t* buffer, size_t length);
00018 };
00019
00020 #endif

```

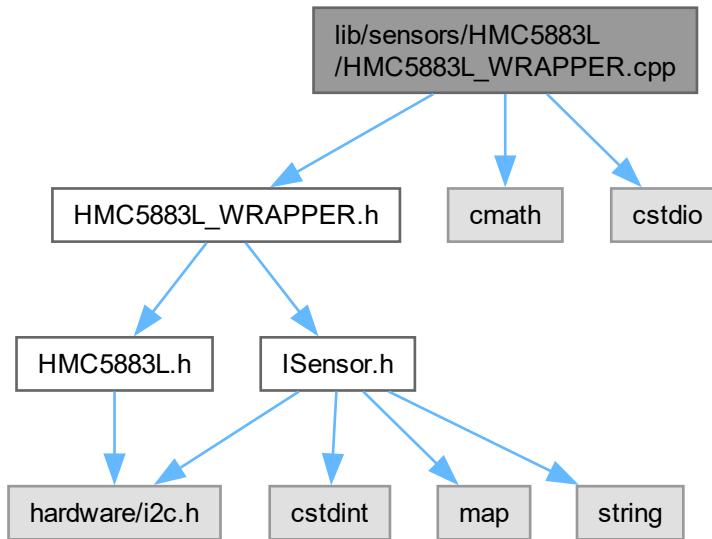
## 8.89 lib/sensors/HMC5883L/HMC5883L\_WRAPPER.cpp File Reference

```

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

```

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



## 8.90 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::readData(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::isInitialized() const {
00036 return initialized;

```

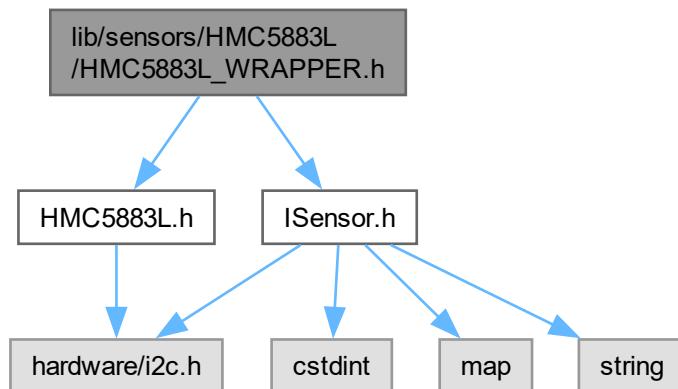
```

00037 }
00038
00039 SensorType HMC5883LWrapper::getType() 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 }

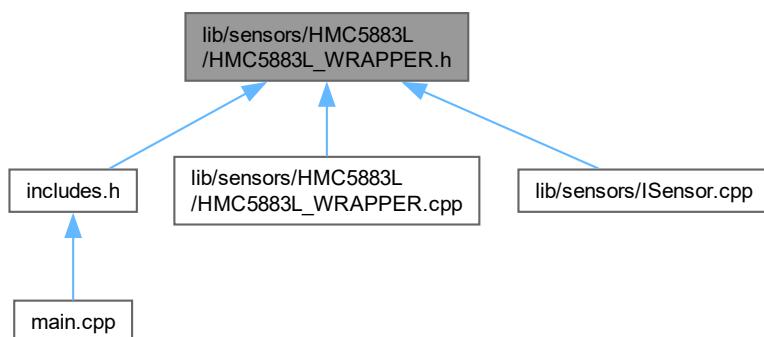
```

## 8.91 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](#)

## 8.92 HMC5883L\_WRAPPER.h

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

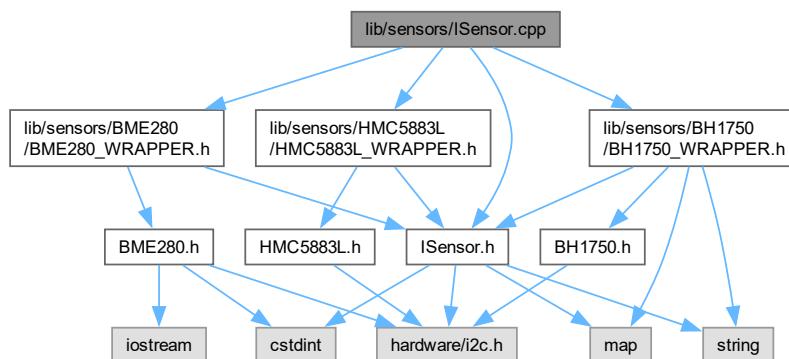
```
00001 #ifndef HMC5883L_WRAPPER_H
00002 #define HMC5883L_WRAPPER_H
00003
00004 #include "ISensor.h"
00005 #include "HMC5883L.h"
00006
00007 class HMC5883LWrapper : public ISensor {
00008 public:
00009 HMC5883LWrapper(i2c_inst_t* i2c);
00010 bool init() override;
00011 float readData(SensorDataTypeIdentifier type) override;
00012 bool isInitialized() const override;
00013 SensorType getType() 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
```

## 8.93 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:



### 8.93.1 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](#).

## 8.94 ISensor.cpp

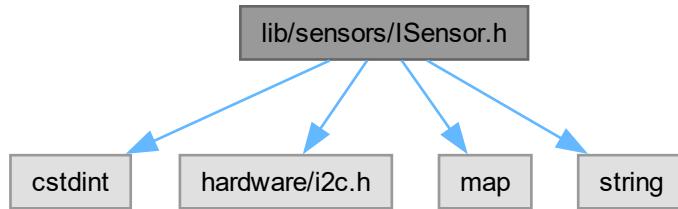
[Go to the documentation of this file.](#)

```
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::getInstance() {
00024 static SensorWrapper instance;
00025 return instance;
00026 }
00027
00031 SensorWrapper::SensorWrapper() = default;
00032
00039 bool SensorWrapper::initSensor(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::configureSensor(SensorType type, const std::map<std::string, std::string>& config)
{
00064 auto it = sensors.find(type);
00065 if (it != sensors.end() && it->second->isInitialized()) {
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::readSensorData(SensorType sensorType, SensorDataTypeIdentifier dataType) {
00079 auto it = sensors.find(sensorType);
00080 if (it != sensors.end() && it->second->isInitialized()) {
00081 return it->second->readData(dataType);
00082 }
00083 return 0.0f;
00084 }
```

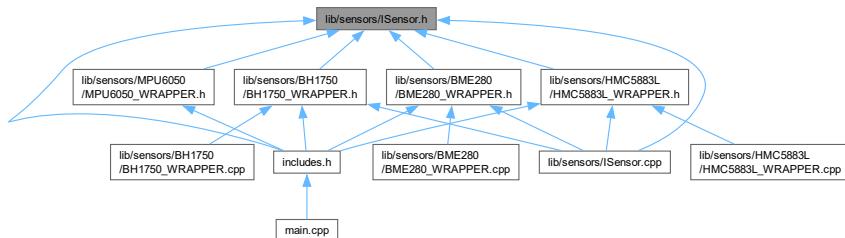
## 8.95 lib/sensors/ISensor.h File Reference

```
#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` }

### 8.95.1 Enumeration Type Documentation

#### 8.95.1.1 SensorType

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

## Enumerator

|              |  |
|--------------|--|
| LIGHT        |  |
| ENVIRONMENT  |  |
| MAGNETOMETER |  |
| IMU          |  |

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

**8.95.1.2 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](#).

**8.96 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 readData(SensorDataTypeIdentifier type) = 0;
00038 virtual bool isInitialized() const = 0;
00039 virtual SensorType getType() 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& getInstance();
00046 bool initSensor(SensorType type, i2c_inst_t* i2c = nullptr);
00047 bool configureSensor(SensorType type, const std::map<std::string, std::string>& config);
00048 float readSensorData(SensorType sensorType, SensorDataTypeIdentifier dataType);
00049
00050 private:
00051 std::map<SensorType, ISensor*> sensors;
00052 SensorWrapper();
00053 };
00054
00055 #endif // ISENSOR_H
```

## 8.97 lib/sensors/MPU6050/MPU6050.cpp File Reference

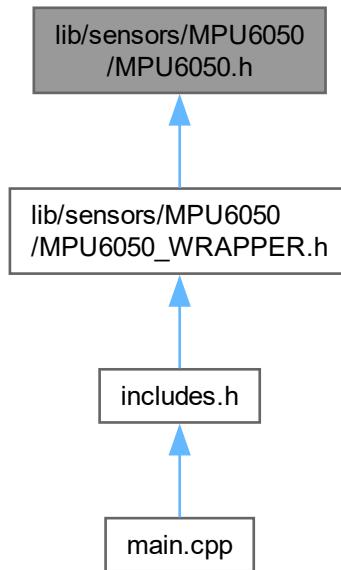
## 8.98 MPU6050.cpp

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

```
00001
```

## 8.99 lib/sensors/MPU6050/MPU6050.h File Reference

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



## 8.100 MPU6050.h

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

00001

## 8.101 lib/sensors/MPU6050/MPU6050\_WRAPPER.cpp File Reference

## 8.102 MPU6050\_WRAPPER.cpp

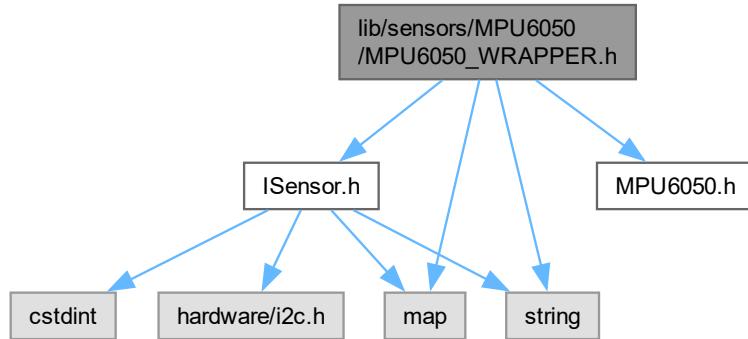
[Go to the documentation of this file.](#)

00001

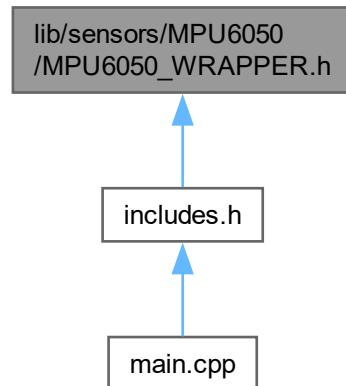
## 8.103 lib/sensors/MPU6050/MPU6050\_WRAPPER.h File Reference

```
#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](#)

## 8.104 MPU6050\_WRAPPER.h

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

```
00001 #ifndef BH1750_WRAPPER_H
00002 #define BH1750_WRAPPER_H
00003
```

```

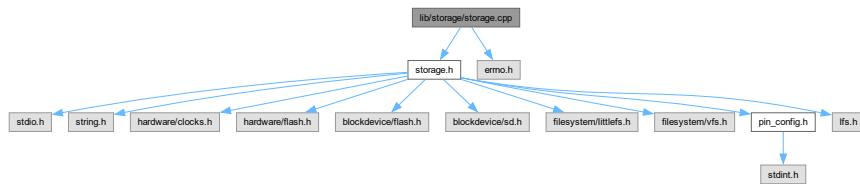
00004 #include "ISensor.h"
00005 #include "MPU6050.h"
00006 #include <map>
00007 #include <string>
00008
00009 class MPU6050Wrapper : public ISensor {
00010 private:
00011 MPU6050 sensor;
00012 bool initialized = false;
00013
00014 public:
00015 MPU6050Wrapper();
00016
00017 bool init() override;
00018 float readData(SensorDataTypeIdentifier type) override;
00019 bool isInitialized() const override;
00020 SensorType getType() const override;
00021
00022 bool configure(const std::map<std::string, std::string>& config);
00023 };
00024
00025 #endif

```

## 8.105 lib/storage/storage.cpp File Reference

Implements file system operations for the Kabisat firmware.

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



## Functions

- **bool fs\_init (void)**  
*Initializes the file system on the SD card.*
- **FileHandle fs\_open\_file (const char \*filename, const char \*mode)**  
*Opens a file.*
- **ssize\_t fs\_write\_file (FileHandle &handle, const void \*buffer, size\_t size)**  
*Writes data to a file.*
- **ssize\_t fs\_read\_file (FileHandle &handle, void \*buffer, size\_t size)**  
*Reads data from a file.*
- **bool fs\_close\_file (FileHandle &handle)**  
*Closes a file.*
- **bool fs\_file\_exists (const char \*filename)**  
*Checks if a file exists.*

## 8.105.1 Detailed Description

Implements file system operations for the KubitSat firmware.

This file contains functions for initializing the file system, opening, writing, reading, and closing files.

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

## 8.105.2 Function Documentation

### 8.105.2.1 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 [23](#) of file [storage.cpp](#).

Here is the caller graph for this function:



### 8.105.2.2 fs\_open\_file()

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

Opens a file.

#### Parameters

|                 |                                                           |
|-----------------|-----------------------------------------------------------|
| <i>filename</i> | The name of the file to open.                             |
| <i>mode</i>     | The mode in which to open the file (e.g., "r", "w", "a"). |

#### Returns

A [FileHandle](#) structure containing the file descriptor and a flag indicating if the file is open.

Opens the specified file with the given mode. Converts the mode string to appropriate flags for the open system call.

Definition at line [60](#) of file [storage.cpp](#).

### 8.105.2.3 fs\_write\_file()

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

Writes data to a file.

#### Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>handle</i> | A reference to the <a href="#">FileHandle</a> structure for the file. |
| <i>buffer</i> | A pointer to the buffer containing the data to write.                 |
| <i>size</i>   | The number of bytes to write.                                         |

#### Returns

The number of bytes written, or -1 on error.

Writes data from the provided buffer to the file associated with the given [FileHandle](#).

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

### 8.105.2.4 fs\_read\_file()

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

Reads data from a file.

#### Parameters

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>handle</i> | A reference to the <a href="#">FileHandle</a> structure for the file. |
| <i>buffer</i> | A pointer to the buffer to store the read data.                       |
| <i>size</i>   | The number of bytes to read.                                          |

#### Returns

The number of bytes read, or -1 on error.

Reads data from the file associated with the given [FileHandle](#) into the provided buffer.

Definition at line 111 of file [storage.cpp](#).

### 8.105.2.5 fs\_close\_file()

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

Closes a file.

**Parameters**

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>handle</i> | A reference to the <a href="#">FileHandle</a> structure for the file. |
|---------------|-----------------------------------------------------------------------|

**Returns**

True if the file was closed successfully, false otherwise.

Closes the file associated with the given [FileHandle](#).

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

**8.105.2.6 fs\_file\_exists()**

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

Checks if a file exists.

**Parameters**

|                 |                                |
|-----------------|--------------------------------|
| <i>filename</i> | The name of the file to check. |
|-----------------|--------------------------------|

**Returns**

True if the file exists, false otherwise.

Checks if a file with the given name exists in the file system.

Definition at line 153 of file [storage.cpp](#).

**8.106 storage.cpp**

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

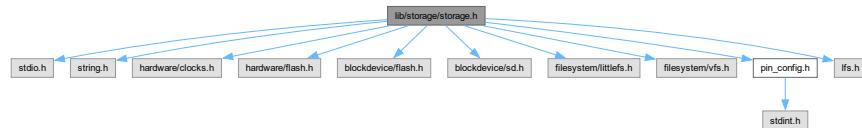
```
00001 /*
00002 * Copyright 2024, Hiroyuki OYAMA. All rights reserved.
00003 *
00004 * SPDX-License-Identifier: BSD-3-Clause
00005 */
00006 #include "storage.h"
00007 #include "errno.h"
00008
00009
00016
00023 bool fs_init(void) {
00024 printf("fs_init littlefs on SD card\n");
00025 blockdevice_t *sd = blockdevice_sd_create(SD_SPI_PORT,
00026 SD_MOSI_PIN,
00027 SD_MISO_PIN,
00028 SD_SCK_PIN,
00029 SD_CS_PIN,
00030 24 * MHZ,
00031 false);
00032 filesystem_t *littlefs = filesystem_littlefs_create(500, 16);
00033 int err = fs_mount("/", littlefs, sd);
00034 if (err == -1) {
00035 printf("format / with littlefs\n");
00036 err = fs_format(littlefs, sd);
00037 if (err == -1) {
```

```
00038 printf("fs_format error: %s", strerror(errno));
00039 return false;
00040 }
00041 err = fs_mount("/", littlefs, sd);
00042 if (err == -1) {
00043 printf("fs_mount error: %s", strerror(errno));
00044 return false;
00045 }
00046 }
00047
00048 return true;
00049 }
00050
00051
00060 FileHandle fs_open_file(const char* filename, const char* mode) {
00061 FileHandle handle = {-1, false};
00062
00063 // Convert mode string to flags
00064 int flags = 0;
00065 if (strchr(mode, 'r')) flags |= O_RDONLY;
00066 if (strchr(mode, 'w')) flags |= O_WRONLY | O_CREAT | O_TRUNC;
00067 if (strchr(mode, 'a')) flags |= O_WRONLY | O_CREAT | O_APPEND;
00068 if (strchr(mode, '+')) flags = O_RDWR | (flags & ~O_RDONLY | O_WRONLY);
00069
00070 // Open the file
00071 handle.fd = open(filename, flags, 0666);
00072 if (handle.fd >= 0) {
00073 handle.is_open = true;
00074 } else {
00075 printf("Failed to open file %s: %s\n", filename, strerror(errno));
00076 }
00077
00078 return handle;
00079 }
00080
00081
00090 ssize_t fs_write_file(FileHandle& handle, const void* buffer, size_t size) {
00091 if (!handle.is_open) {
00092 return -1;
00093 }
00094
00095 ssize_t written = write(handle.fd, buffer, size);
00096 if (written < 0) {
00097 printf("Write failed: %s\n", strerror(errno));
00098 }
00099 return written;
00100 }
00101
00102
00111 ssize_t fs_read_file(FileHandle& handle, void* buffer, size_t size) {
00112 if (!handle.is_open) {
00113 return -1;
00114 }
00115
00116 ssize_t bytes_read = read(handle.fd, buffer, size);
00117 if (bytes_read < 0) {
00118 printf("Read failed: %s\n", strerror(errno));
00119 }
00120 return bytes_read;
00121 }
00122
00123
00130 bool fs_close_file(FileHandle& handle) {
00131 if (!handle.is_open) {
00132 return false;
00133 }
00134
00135 int result = close(handle.fd);
00136 if (result == 0) {
00137 handle.is_open = false;
00138 handle.fd = -1;
00139 return true;
00140 } else {
00141 printf("Close failed: %s\n", strerror(errno));
00142 return false;
00143 }
00144 }
00145
00146
00153 bool fs_file_exists(const char* filename) {
00154 struct stat st;
00155 return (stat(filename, &st) == 0);
00156 }
```

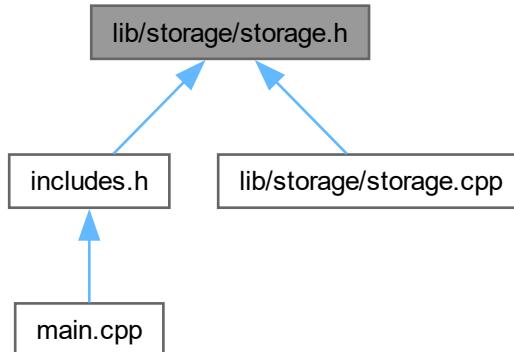
## 8.107 lib/storage/storage.h File Reference

```
#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 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)
 

*Opens a file.*

- `ssize_t fs_write_file (FileHandle &handle, const void *buffer, size_t size)`  
*Writes data to a file.*
- `ssize_t fs_read_file (FileHandle &handle, void *buffer, size_t size)`  
*Reads data from a file.*
- `bool fs_close_file (FileHandle &handle)`  
*Closes a file.*
- `bool fs_file_exists (const char *filename)`  
*Checks if a file exists.*

## 8.107.1 Function Documentation

### 8.107.1.1 `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 23 of file [storage.cpp](#).

Here is the caller graph for this function:



### 8.107.1.2 `fs_open_file()`

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

Opens a file.

#### Parameters

|                       |                                                           |
|-----------------------|-----------------------------------------------------------|
| <code>filename</code> | The name of the file to open.                             |
| <code>mode</code>     | The mode in which to open the file (e.g., "r", "w", "a"). |

#### Returns

A `FileHandle` structure containing the file descriptor and a flag indicating if the file is open.

Opens the specified file with the given mode. Converts the mode string to appropriate flags for the `open` system call.

Definition at line 60 of file [storage.cpp](#).

### 8.107.1.3 `fs_write_file()`

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

Writes data to a file.

#### Parameters

|                     |                                                                       |
|---------------------|-----------------------------------------------------------------------|
| <code>handle</code> | A reference to the <a href="#">FileHandle</a> structure for the file. |
| <code>buffer</code> | A pointer to the buffer containing the data to write.                 |
| <code>size</code>   | The number of bytes to write.                                         |

#### Returns

The number of bytes written, or -1 on error.

Writes data from the provided buffer to the file associated with the given [FileHandle](#).

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

### 8.107.1.4 `fs_read_file()`

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

Reads data from a file.

#### Parameters

|                     |                                                                       |
|---------------------|-----------------------------------------------------------------------|
| <code>handle</code> | A reference to the <a href="#">FileHandle</a> structure for the file. |
| <code>buffer</code> | A pointer to the buffer to store the read data.                       |
| <code>size</code>   | The number of bytes to read.                                          |

#### Returns

The number of bytes read, or -1 on error.

Reads data from the file associated with the given [FileHandle](#) into the provided buffer.

Definition at line 111 of file [storage.cpp](#).

### 8.107.1.5 `fs_close_file()`

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

Closes a file.

**Parameters**

|               |                                                                       |
|---------------|-----------------------------------------------------------------------|
| <i>handle</i> | A reference to the <a href="#">FileHandle</a> structure for the file. |
|---------------|-----------------------------------------------------------------------|

**Returns**

True if the file was closed successfully, false otherwise.

Closes the file associated with the given [FileHandle](#).

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

**8.107.1.6 fs\_file\_exists()**

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

Checks if a file exists.

**Parameters**

|                 |                                |
|-----------------|--------------------------------|
| <i>filename</i> | The name of the file to check. |
|-----------------|--------------------------------|

**Returns**

True if the file exists, false otherwise.

Checks if a file with the given name exists in the file system.

Definition at line 153 of file [storage.cpp](#).

**8.108 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
00015 struct FileHandle {
00016 int fd;
00017 bool is_open;
00018 };
00019
00020 bool fs_init(void);
00021 FileHandle fs_open_file(const char* filename, const char* mode);
00022 ssize_t fs_write_file(FileHandle& handle, const void* buffer, size_t size);
00023 ssize_t fs_read_file(FileHandle& handle, void* buffer, size_t size);
00024 bool fs_close_file(FileHandle& handle);
00025 bool fs_file_exists(const char* filename);
```

```

00026
00027 #endif
00028
00029 // void example_file_operations() {
00030 // // Open a file for writing
00031 // FileHandle log_file = fs_open_file("/log.txt", "w");
00032 // if (!log_file.is_open) {
00033 // uartPrint("Failed to open log file");
00034 // return;
00035 // }
00036
00037 // // Write some data
00038 // const char* message = "Hello, World!\n";
00039 // ssize_t written = fs_write_file(log_file, message, strlen(message));
00040 // if (written < 0) {
00041 // uartPrint("Failed to write to log file");
00042 // }
00043
00044 // // Close the file
00045 // fs_close_file(log_file);
00046
00047 // // Open file for reading
00048 // log_file = fs_open_file("/log.txt", "r");
00049 // if (!log_file.is_open) {
00050 // uartPrint("Failed to open log file for reading");
00051 // return;
00052 // }
00053
00054 // // Read the data
00055 // char buffer[128];
00056 // ssize_t bytes_read = fs_read_file(log_file, buffer, sizeof(buffer) - 1);
00057 // if (bytes_read > 0) {
00058 // buffer[bytes_read] = '\0'; // Null terminate the string
00059 // uartPrint(buffer);
00060 // }
00061
00062 // // Close the file
00063 // fs_close_file(log_file);
00064 //

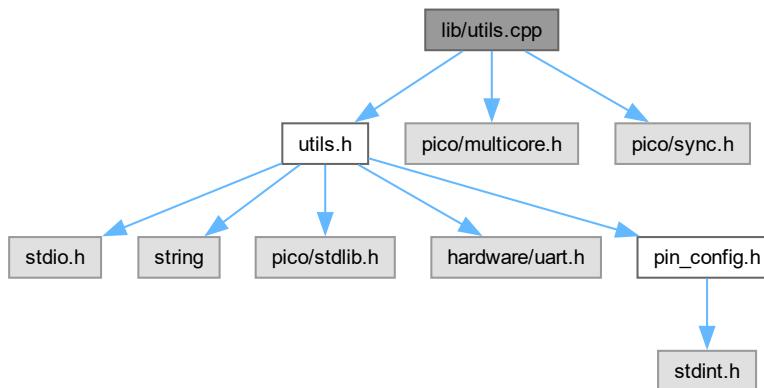
```

## 8.109 lib/utils.cpp File Reference

Implements utility functions for the Kabisat firmware.

```
#include "utils.h"
#include "pico/multicore.h"
#include "pico/sync.h"
```

Include dependency graph for utils.cpp:



## Macros

- `#define LOG_FILENAME "log.txt"`

## Functions

- `void uartPrint (const std::string &msg, 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`

### 8.109.1 Detailed Description

Implements utility functions for the Kabisat firmware.

This file contains various utility functions, including UART printing with timestamps, and CRC16 calculation.

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

### 8.109.2 Macro Definition Documentation

#### 8.109.2.1 LOG\_FILENAME

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

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

### 8.109.3 Function Documentation

#### 8.109.3.1 uartPrint()

```
void uartPrint (
 const std::string & msg,
 bool logToFile,
 uart_inst_t * uart)
```

Prints a message to the UART with a timestamp and core number.

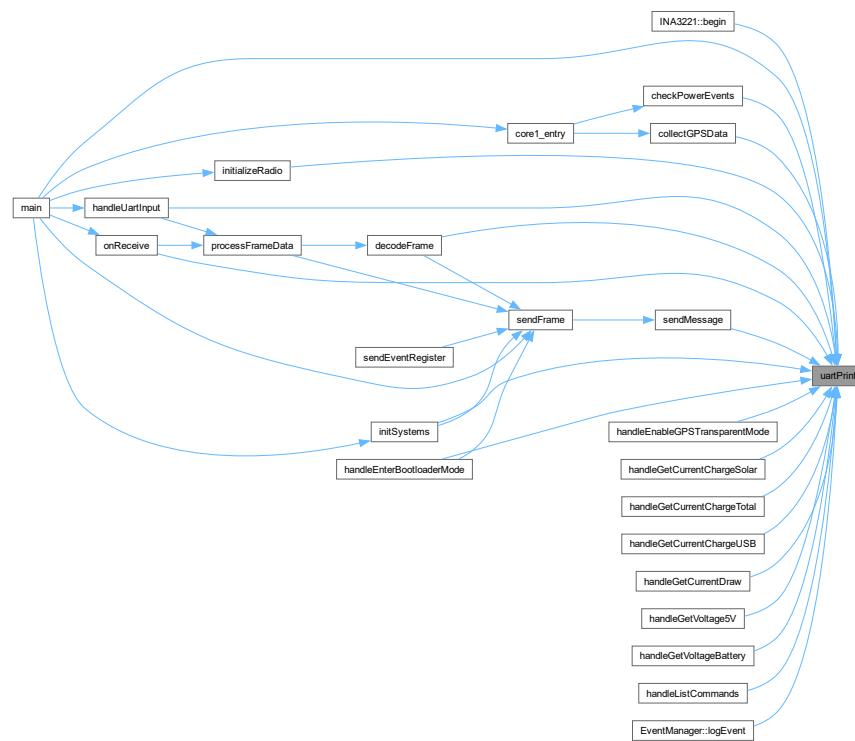
##### Parameters

|                        |                                                                                     |
|------------------------|-------------------------------------------------------------------------------------|
| <code>msg</code>       | The message to print.                                                               |
| <code>logToFile</code> | A flag indicating whether to log the message to a file (currently not implemented). |
| <code>uart</code>      | 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 26 of file [utils.cpp](#).

Here is the caller graph for this function:



### 8.109.3.2 crc16()

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

Calculates the CRC16 checksum of the given data.

#### 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 53 of file [utils.cpp](#).

## 8.109.4 Variable Documentation

### 8.109.4.1 uart\_mutex

```
mutex_t uart_mutex [static]
```

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

## 8.110 utils.cpp

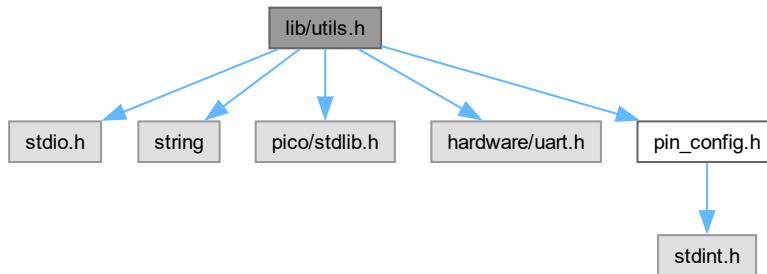
[Go to the documentation of this file.](#)

```
00001 #include "utils.h"
00002 #include "pico/multicore.h"
00003 #include "pico/sync.h"
00004
00005 #define LOG_FILENAME "log.txt"
00006
00007 static mutex_t uart_mutex;
00008
00009
00016
00017
00026 void uartPrint(const std::string& msg, bool logToFile, uart_inst_t* uart) {
00027 static bool mutex_initiated = false;
00028 if (!mutex_initiated) {
00029 mutex_init(&uart_mutex);
00030 mutex_initiated = true;
00031 }
00032
00033 uint32_t timestamp = to_ms_since_boot(get_absolute_time());
00034 uint core_num = get_core_num();
00035 std::string msgToSend = "[" + std::to_string(timestamp) + "ms] - Core " + std::to_string(core_num)
+ ":" + msg + "\r\n";
00036
00037 if (logToFile) {
00038 logToFile = !logToFile;
00039 }
00040 mutex_enter_blocking(&uart_mutex);
00041 uart_puts(uart, msgToSend.c_str());
00042 mutex_exit(&uart_mutex);
00043 }
00044
00045
00053 uint16_t crc16(const uint8_t *data, size_t length) {
00054 uint16_t crc = 0xFFFF;
00055 for (size_t i = 0; i < length; i++) {
00056 crc ^= data[i];
00057 for (int j = 0; j < 8; j++) {
00058 if (crc & 0x0001) {
00059 crc = (crc >> 1) ^ 0xA001;
00060 } else {
00061 crc >>= 1;
00062 }
00063 }
00064 }
00065 return crc;
00066 }
```

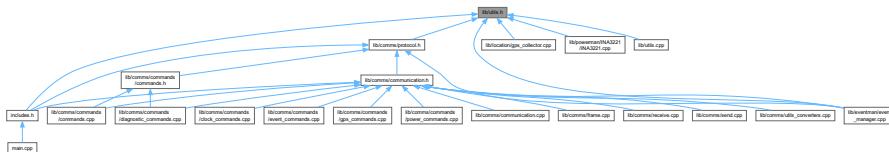
## 8.111 lib/utils.h File Reference

```
#include <stdio.h>
#include <string>
#include "pico/stdlib.h"
#include "hardware/uart.h"
```

```
#include "pin_config.h"
Include dependency graph for utils.h:
```



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



## Functions

- void `uartPrint` (const std::string &`msg`, bool `logToFile=false`, uart\_inst\_t \*`uart=DEBUG_UART_PORT`)
   
*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.*

### 8.111.1 Function Documentation

#### 8.111.1.1 `uartPrint()`

```
void uartPrint (
 const std::string & msg,
 bool logToFile,
 uart_inst_t * uart)
```

Prints a message to the UART with a timestamp and core number.

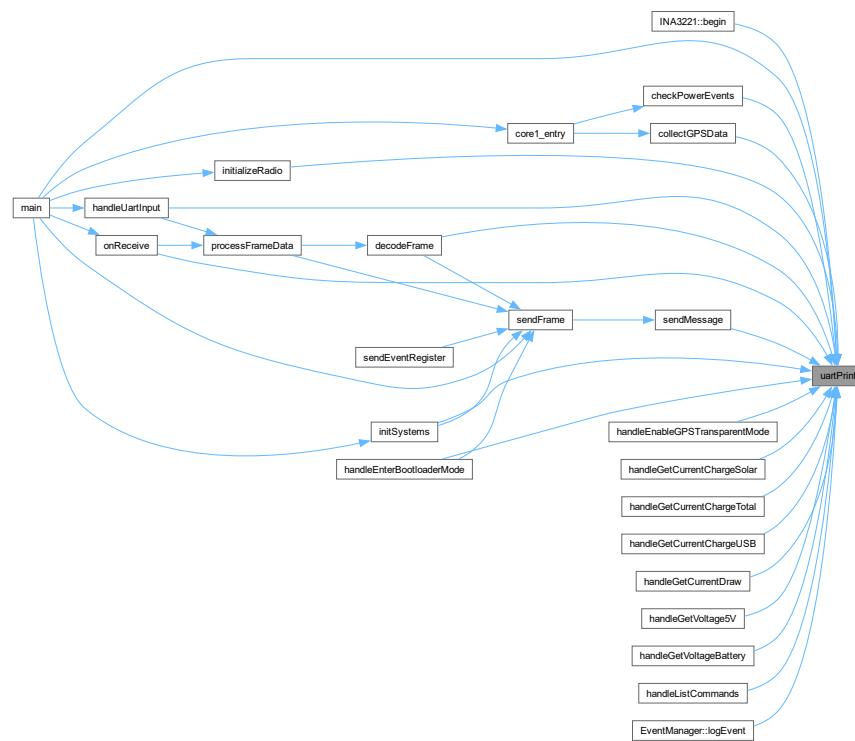
##### Parameters

|                        |                                                                                     |
|------------------------|-------------------------------------------------------------------------------------|
| <code>msg</code>       | The message to print.                                                               |
| <code>logToFile</code> | A flag indicating whether to log the message to a file (currently not implemented). |
| <code>uart</code>      | 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 26 of file [utils.cpp](#).

Here is the caller graph for this function:



### 8.111.1.2 crc16()

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

Calculates the CRC16 checksum of the given data.

#### 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 53 of file [utils.cpp](#).

## 8.112 utils.h

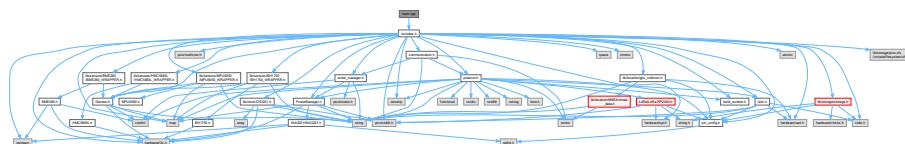
[Go to the documentation of this file.](#)

```
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003
00004 #include <stdio.h>
00005 #include <string>
00006 #include "pico/stdlib.h"
00007 #include "hardware/uart.h"
00008 #include "pin_config.h"
00009
00010 void uartPrint(const std::string& msg, bool logToFile = false, uart_inst_t* uart = DEBUG_UART_PORT);
00011 uint16_t crl16(const uint8_t *data, size_t length);
00012
00013 #endif
```

## 8.113 main.cpp File Reference

```
#include "includes.h"
```

Include dependency graph for main.cpp:



## Macros

- #define LOG\_FILENAME "log.txt"

## Functions

- void `core1_entry ()`
  - bool `initSystems ()`
  - void `loggingRoutine ()`
  - int `main ()`

## Variables

- PowerManager powerManager (MAIN\_I2C\_PORT)
  - DS3231 systemClock (MAIN\_I2C\_PORT)
  - char buffer [BUFFER\_SIZE]
  - int bufferIndex = 0

### 8.113.1 Macro Definition Documentation

### **8.113.1.1 LOG FILENAME**

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

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

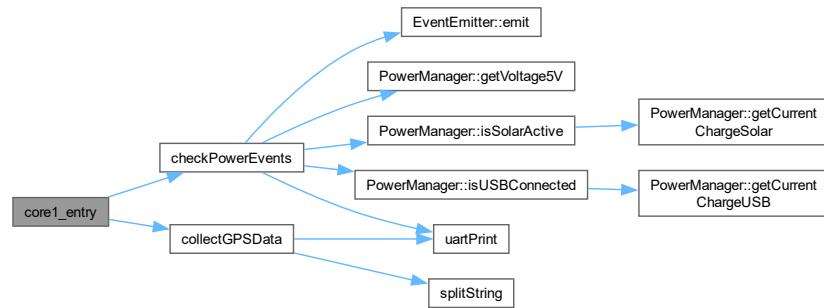
## 8.113.2 Function Documentation

### 8.113.2.1 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:

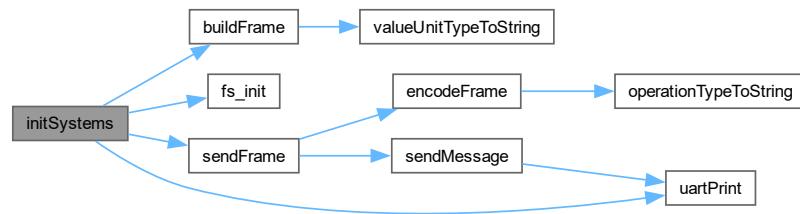


### 8.113.2.2 initSystems()

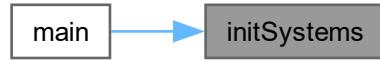
```
bool initSystems ()
```

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

Here is the call graph for this function:



Here is the caller graph for this function:



### 8.113.2.3 loggingRoutine()

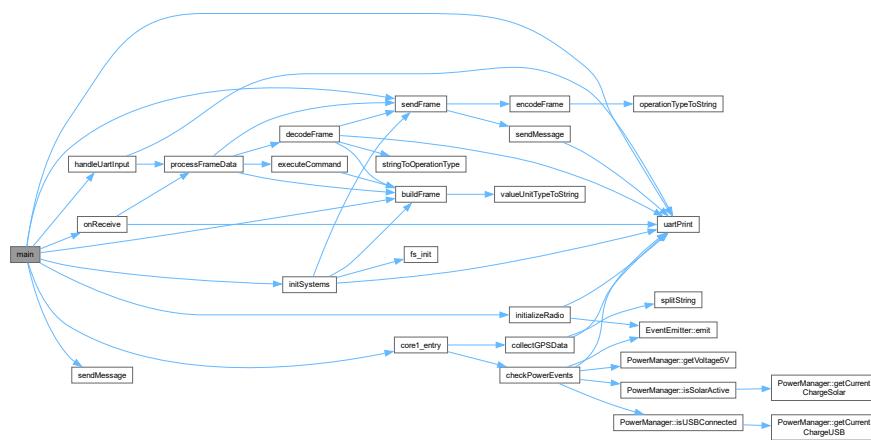
```
void loggingRoutine ()
```

### 8.113.2.4 main()

```
int main ()
```

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

Here is the call graph for this function:



## 8.113.3 Variable Documentation

### 8.113.3.1 powerManager

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

### 8.113.3.2 systemClock

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

### 8.113.3.3 buffer

```
char buffer[BUFFER_SIZE]
```

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

### 8.113.3.4 bufferIndex

```
int bufferIndex = 0
```

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

## 8.114 main.cpp

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

```
00001 #include "includes.h"
00002
00003 #define LOG_FILENAME "log.txt"
00004
00005 PowerManager powerManager(MAIN_I2C_PORT);
00006 DS3231 systemClock(MAIN_I2C_PORT);
00007
00008 char buffer[BUFFER_SIZE];
00009 int bufferIndex = 0;
00010
00011 void core1_entry() {
00012 while (true) {
00013 collectGPSData();
00014 checkPowerEvents(powerManager);
00015 }
00016 }
00017
00018 bool initSystems() {
00019 stdio_init_all();
00020
00021 uart_init(DEBUG_UART_PORT, DEBUG_UART_BAUD_RATE);
00022 gpio_set_function(DEBUG_UART_TX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_TX_PIN));
00023 gpio_set_function(DEBUG_UART_RX_PIN, UART_FUNCSEL_NUM(DEBUG_UART_PORT, DEBUG_UART_RX_PIN));
00024
00025 gpio_init(PICO_DEFAULT_LED_PIN);
00026 gpio_set_dir(PICO_DEFAULT_LED_PIN, GPIO_OUT);
00027
00028 i2c_init(MAIN_I2C_PORT, 400 * 1000);
00029 gpio_set_function(MAIN_I2C_SCL_PIN, GPIO_FUNC_I2C);
00030 gpio_set_function(MAIN_I2C_SDA_PIN, GPIO_FUNC_I2C);
00031 gpio_pull_up(MAIN_I2C_SCL_PIN);
00032 gpio_pull_up(MAIN_I2C_SDA_PIN);
00033
00034 uart_init(GPS_UART_PORT, GPS_UART_BAUD_RATE);
00035 gpio_set_function(GPS_UART_TX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_TX_PIN));
00036 gpio_set_function(GPS_UART_RX_PIN, UART_FUNCSEL_NUM(GPS_UART_PORT, GPS_UART_RX_PIN));
00037
00038 if (true)
00039 {
00040 gpio_init(GPS_POWER_ENABLE_PIN);
00041 gpio_set_dir(GPS_POWER_ENABLE_PIN, GPIO_OUT);
00042 gpio_put(GPS_POWER_ENABLE_PIN, 1);
00043 }
00044
00045 // @todo[critical] Test sd card working
00046 bool sdInitDone = fs_init();
00047 uartPrint("SD card init: " + std::to_string(sdInitDone));
```

```
00048 std::string bootString = "System init completed @ " +
00049 std::to_string(to_ms_since_boot(get_absolute_time())) + " ms";
00050 uartPrint(bootString);
00051
00052 Frame boot = buildFrame(ExecutionResult::INFO, 0, 0, "HELLO");
00053 sendFrame/boot();
00054
00055 return true;
00056 }
00057 void loggingRoutine();
00058
00059 int main()
00060 {
00061 initSystems();
00062 multicore_launch_core1(core1_entry);
00063
00064 systemClock.setTime(0, 41, 20, 4, 14, 11, 2024);
00065 gpio_put(PICO_DEFAULT_LED_PIN, 1);
00066
00067 if (powerManager.initialize())
00068 {
00069 std::map<std::string, std::string> powerConfig = {
00070 {"operating_mode", "continuous"},
00071 {"averaging_mode", "16"}, };
00072 };
00073 powerManager.configure(powerConfig);
00074 }
00075 gpio_put(PICO_DEFAULT_LED_PIN, 0);
00076
00077 bool radioInitSuccess = false;
00078
00079 radioInitSuccess = initializeRadio();
00080 gpio_put(PICO_DEFAULT_LED_PIN, 1);
00081
00082 if (radioInitSuccess)
00083 {
00084 sendMessage("System initialized successfully!");
00085 }
00086
00087 gpio_put(PICO_DEFAULT_LED_PIN, 0);
00088
00089 uartPrint("This message will only be printed to UART.");
00090 // uartPrint("This message will be printed to UART and logged to Core 1.", true);
00091
00092 for (int i = 5; i > 0; --i)
00093 {
00094 std::string intro = "Main loop starts in " + std::to_string(i) + " seconds...";
00095 uartPrint/intro();
00096 gpio_put(PICO_DEFAULT_LED_PIN, (i%2==0));
00097 sleep_ms(1000);
00098 }
00099 gpio_put(PICO_DEFAULT_LED_PIN, 1);
00100
00101 Frame boot = buildFrame(ExecutionResult::INFO, 0, 0, "START");
00102 sendFrame/boot();
00103
00104 while (true)
00105 {
00106 int packetSize = LoRa.parsePacket();
00107 if (packetSize)
00108 {
00109 onReceive(packetSize);
00110 }
00111
00112 //acquireLock();
00113 //float voltage = sharedData.voltage5V;
00114 //releaseLock();
00115
00116 //std::string voltageReading = "Core 0: voltage from common data structure: " +
00117 std::to_string(voltage);
00118 //uartPrint(voltageReading.c_str());
00119
00120 //collectGPSData();
00121 handleUartInput();
00122 }
00123
00124 return 0;
00125 }
00126 // BH1750 0X23
00127 // INA3221 0X40
00128 // BME280 0X76
00129 // DS3231 0X86
```