SimTab

Generated by Doxygen 1.7.6.1

Tue Oct 22 2013 21:14:11

Contents

1 Module Index

1.1 Modules

Here is a list of all modules:

Utitlity classes for threading with pthread	**6**
Classes related to communication with Pololu MinIMU	**7**
Classes related to communication with MicroStrain 3DM-GX3	**8**
Classes related to controlling the motors	**10**

2 Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

3 Class Index

3.1 Class Hierarchy

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

4 Class Index

4.1 Class List

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

5 File Index

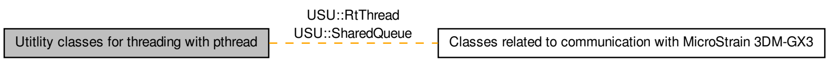
5.1 File List

Here is a list of all files with brief descriptions:

6 Module Documentation

6.1 Utitlity classes for threading with pthread

Collaboration diagram for Utitlity classes for threading with pthread:


Utitlity classes for threading with pthread
USU::RtThread
USU::SharedQueue
Classes related to communication with MicroStrain 3DM-GX3

Classes

- class USU::Lock

  Wrapper class for pthread mutexes.
- class USU::ScopedLock

  Provides a helper class for Scoped Mutexes.
- class USU::PeriodicRtThread

  TODO: Make some proper exceptions.
- class USU::RtThread

  Abstract wrapper class for the pthread library with RT-priority.
- class USU::Semaphore

  Wrapper class for semaphores.

- class USU::SharedQueue< T >

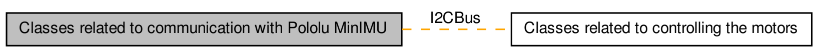
  Wrapper class to make std::queue thread safe.

6.1.1 Detailed Description

Yadsjflsfjlk yadadada dadadljfsfj

6.2 Classes related to communication with Pololu MinIMU

Collaboration diagram for Classes related to communication with Pololu MinIMU:


Classes related to communication with Pololu MinIMU
I2CBus
Classes related to controlling the motors

Classes

- class I2CBus

  Wrapper class for I2C-bus communication.
- class IMU

  Virtual base class for IMU.
- class L3G

  Class to manage the communication to the L3G gyroscope via the I2C-bus.
- class LSM303

  Class to manage communication to the LSM303 compass via the I2C-bus.
- class USU::MinImu

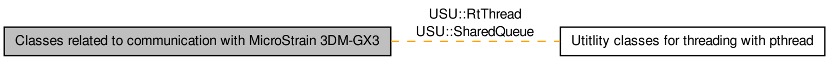
  Class to manage the communication to the Pololu MinIMU9.

6.2.1 Detailed Description

TODO: Write something here

6.3 Classes related to communication with MicroStrain 3DM-GX3

Collaboration diagram for Classes related to communication with MicroStrain 3DM-GX3:


Classes related to communication with MicroStrain 3DM-GX3
USU::RtThread
USU::SharedQueue
Utitlity classes for threading with pthread

Classes

- class USU::GX3Packet

  Abstract base class for received packets.
- class USU::RawAccAng

  Representation for receiving (raw) acceleration & angular rate packets.
- class USU::AccAngMag

  Representation for receiving acceleration, angular rate and magnetometer packets.
- class USU::Quaternion

  Representation for receiving the Quaternion representation from the IMU.
- class USU::AccAngMagOrientationMat

  Representation for packets containing the 3 sensor vectors and orientation matrix This class can be used with the commands which return 3 Vectors and a 3x3 Matrix. The units are:
- class USU::GX3Command

  Base class for commands send to the 3DM-GX3-25.
- class USU::SetCountinuousMode

  Represents the "Set continuous mode" command.
- class USU::SamplingSettings

  Represents the "Sampling Settings" command.
- class USU::RtThread

  Abstract wrapper class for the pthread library with RT-priority.
- class USU::SharedQueue< T >

  Wrapper class to make std::queue thread safe.

Typedefs

- typedef std::shared_ptr < GX3Packet > USU::packet_ptr

  Represents the Thread class for communication with the 3DM-GX3-25.

6.3.1 Detailed Description

TODO: Write something here

6.3.2 Typedef Documentation

6.3.2.1 typedef std::shared_ptr<GX3Packet> USU::packet_ptr

Represents the Thread class for communication with the 3DM-GX3-25.

The class is derived from RtThread. It initializes the serial interface to the 3DM and sets the sampling settings. Finally it starts the continuous mode and polls the serial port for new arrived data. New data is stored in a FIFO queue.

TODO: Use the parent class for the package instead to make it more generic.

3

Shared pointer for packages

In order to store any kind of a GX3Package in the queue a pointer must be used. Shared pointer is used to avoid memory leaks.

Definition at line 44 of file gx3communicator.h.

6.4 Classes related to controlling the motors

Collaboration diagram for Classes related to controlling the motors:

Classes related to controlling the motors
I2CBus
Classes related to communication with Pololu MinIMU

Classes

- class Beagle_GPIO

  Wrapper class to access the GPIOs of the BeagleBone.
- class cPWM

  Wrapper class to access the PWM-devices of the BeagleBone.
- class I2CBus

  Wrapper class for I2C-bus communication.
- class USU::Max127

  Class representing the MAX127 ADC.
- class USU::Motor

  Class which represents a motor.
- class USU::MotorControl

  Represents the class for motor control.

6.4.1 Detailed Description

TODO: Write something here

7 Namespace Documentation

7.1 USU Namespace Reference

TODO: Make some proper exceptions.

Classes

- class GX3Communicator
- class Lock

  Wrapper class for pthread mutexes.
- class ScopedLock

  Provides a helper class for Scoped Mutexes.
- class MainThread

  Represents the Periodic Thread class for state estimation.
- class Max127

  Class representing the MAX127 ADC.
- class GX3Packet

  Abstract base class for received packets.
- class RawAccAng

  Representation for receiving (raw) acceleration & angular rate packets.
- class AccAngMag

  Representation for receiving acceleration, angular rate and magnetometer packets.
- class Quaternion

  Representation for receiving the Quaternion representation from the IMU.
- class AccAngMagOrientationMat

  Representation for packets containing the 3 sensor vectors and orientation matrix This class can be used with the commands which return 3 Vectors and a 3x3 Matrix. The units are:
- class GX3Command

  Base class for commands send to the 3DM-GX3-25.
- class SetCountinuousMode

  Represents the "Set continuous mode" command.
- class SamplingSettings

  Represents the "Sampling Settings" command.
- class MinImu

  Class to manage the communication to the Pololu MinIMU9.
- class Motor

  Class which represents a motor.
- class MotorControl

  Represents the class for motor control.
- class PeriodicRtThread

  TODO: Make some proper exceptions.

- class RtThread

    Abstract wrapper class for the pthread library with RT-priority.
- class Semaphore

    Wrapper class for semaphores.
- class SharedQueue

    Wrapper class to make std::queue thread safe.

Typedefs

- typedef std::shared_ptr < GX3Packet > packet_ptr

    Represents the Thread class for communication with the 3DM-GX3-25.

Variables

- const uint8_t I2C_ADDRESS = 0b00101000

    I2C-address of the ADC.
- const uint8_t CONTROL_BYTE = 0b10000110

    Template of the control byte.
- const uint8_t SEL0 = 4
- const uint8_t RAW_ACC_ANG = 0xC1
- const uint8_t ACC_ANG = 0xC2
- const uint8_t DELTA_ANGLE_VEL = 0xC3
- const uint8_t SET_CONTINUOUS_MODE = 0xC4
- const uint8_t ORIENTATION_MATRIX = 0xC5
- const uint8_t ORIENTATION_UPDATE_MAT = 0xC6
- const uint8_t MAG_VEC = 0xC7
- const uint8_t ACC_ANG_ORIENTATION_MAT = 0xC8
- const uint8_t WRITE_ACC_BIAS_CORRECTION = 0xC9
- const uint8_t WRITE_GYRO_BIAS_CORRECTION = 0xCA
- const uint8_t ACC_ANG_MAG_VEC = 0xCB
- const uint8_t ACC_ANG_MAG_VEC_ORIENTATION_MAT = 0xCC
- const uint8_t CAPTURE_GYRO_BIAS = 0xCD
- const uint8_t EULER_ANGLES = 0xCE
- const uint8_t EULER_ANGLES_ANG_RATES = 0xCF
- const uint8_t TRANSFER_TO_NONVOL_MEM = 0xD0
- const uint8_t TEMPERATURES = 0xD1
- const uint8_t GYRO_STABIL_ACC_ANG_MAG = 0xD2
- const uint8_t DELTA_ANGLE_VEL_MAG_VEC = 0xD3
- const uint8_t MODE = 0xD4
- const uint8_t MODE_PRESET = 0xD5
- const uint8_t CONTINUOUS_PRESET = 0xD6
- const uint8_t TIMER = 0xD7
- const uint8_t COMM_SETTINGS = 0xD9
- const uint8_t STATIONARY_TEST = 0xDA
- const uint8_t SAMPLING_SETTINGS = 0xDB

- const uint8_t REALIGN_UP_NORTH = 0xDD
- const uint8_t QUATERNION = 0xDF
- const uint8_t WRITE_WORD_EEPROM = 0xE4
- const uint8_t READ_WORD_EEPROM = 0xE5
- const uint8_t READ_FIRMWARE_VER = 0xE9
- const uint8_t READ_DEVICE_ID = 0xEA
- const uint8_t STOP_CONTINUOUS = 0xFA
- const uint8_t FIRMWARE_UPDATE = 0xFD
- const uint8_t DEVICE_RESET = 0xFE

7.1.1 Detailed Description

TODO: Make some proper exceptions.

7.1.2 Variable Documentation

7.1.2.1 const uint8_t USU::ACC_ANG = 0xC2

Acceleration & Angular Rate

Definition at line 30 of file messages.h.

7.1.2.2 const uint8_t USU::ACC_ANG_MAG_VEC = 0xCB

Acceleration, Angular Rate & Magnetometer Vector

Definition at line 39 of file messages.h.

7.1.2.3 const uint8_t USU::ACC_ANG_MAG_VEC_ORIENTATION_MAT = 0xCC

Acceleration, Angular Rate & Magnetometer Vectors & Orientation Matrix

Definition at line 40 of file messages.h.

7.1.2.4 const uint8_t USU::ACC_ANG_ORIENTATION_MAT = 0xC8

Acceleration, Angular Rate & Orientation Matrix

Definition at line 36 of file messages.h.

7.1.2.5 const uint8_t USU::CAPTURE_GYRO_BIAS = 0xCD

Capture Gyro Bias

Definition at line 41 of file messages.h.

7.1.2.6 const uint8_t USU::COMM_SETTINGS = 0xD9

Communications Settings

Definition at line 52 of file messages.h.

7.1.2.7 const uint8_t USU::CONTINUOUS_PRESET = 0xD6

Continuous Preset

Definition at line 50 of file messages.h.

7.1.2.8 const uint8_t USU::CONTROL_BYTE = 0b10000110

Template of the control byte.

The used settings are_

- fullscale range +-5V
- Standby Power-Down mode

The bits for channel selection are set to 0. Send CONTROL_BYTE | (CH<<SEL0) with CH being the desired channel via the I2CBus.

Definition at line 40 of file max127.h.

7.1.2.9 const uint8_t USU::DELTA_ANGLE_VEL = 0xC3

DeltaAngle & DeltaVelocity

Definition at line 31 of file messages.h.

7.1.2.10 const uint8_t USU::DELTA_ANGLE_VEL_MAG_VEC = 0xD3

DeltaAngle & DeltaVelocity & Magnetometer Vectors

Definition at line 47 of file messages.h.

7.1.2.11 const uint8_t USU::DEVICE_RESET = 0xFE

Device Reset (no reply)

Definition at line 63 of file messages.h.

7.1.2.12 const uint8_t USU::EULER_ANGLES = 0xCE

Euler Angles

Definition at line 42 of file messages.h.

7.1.2.13 const uint8_t USU::EULER_ANGLES_ANG_RATES = 0xCF

Euler Angles and Angular Rates

Definition at line 43 of file messages.h.

7.1.2.14 const uint8_t USU::FIRMWARE_UPDATE = 0xFD

Firmware Update (no reply)

Definition at line 62 of file messages.h.

7.1.2.15 const uint8_t USU::GYRO_STABIL_ACC_ANG_MAG = 0xD2

Gyro Stabilized Acceleration, Angular Rate & Magnetometer

Definition at line 46 of file messages.h.

7.1.2.16 const uint8_t USU::I2C_ADDRESS = 0b00101000

I2C-address of the ADC.

It is assumed that the PINs A0-A2 are connected to GND. If the PINs are connected to VCC change accordingly.

Definition at line 27 of file max127.h.

7.1.2.17 const uint8_t USU::MAG_VEC = 0xC7

Magnetometer Vector

Definition at line 35 of file messages.h.

7.1.2.18 const uint8_t USU::MODE = 0xD4

Mode

Definition at line 48 of file messages.h.

7.1.2.19 const uint8_t USU::MODE_PRESET = 0xD5

Mode Preset

Definition at line 49 of file messages.h.

7.1.2.20 const uint8_t USU::ORIENTATION_MATRIX = 0xC5

Orientation Matrix

Definition at line 33 of file messages.h.

7.1.2.21 const uint8_t USU::ORIENTATION_UPDATE_MAT = 0xC6

Orientation Update Matrix

Definition at line 34 of file messages.h.

7.1.2.22 const uint8_t USU::QUATERNION = 0xDF

Quaternion

Definition at line 56 of file messages.h.

7.1.2.23 const uint8_t USU::RAW_ACC_ANG = 0xC1

Raw Accelerometer and Angular Rate Sensor Outputs

Definition at line 29 of file messages.h.

7.1.2.24 const uint8_t USU::READ_DEVICE_ID = 0xEA

Read Device ID String

Definition at line 60 of file messages.h.

7.1.2.25 const uint8_t USU::READ_FIRMWARE_VER = 0xE9

Read Firmware Version Number

Definition at line 59 of file messages.h.

7.1.2.26 const uint8_t USU::READ_WORD_EEPROM = 0xE5

Read Word from EEPROM

Definition at line 58 of file messages.h.

7.1.2.27 const uint8_t USU::REALIGN_UP_NORTH = 0xDD

Realign Up and North

Definition at line 55 of file messages.h.

7.1.2.28 const uint8_t USU::SAMPLING_SETTINGS = 0xDB

Sampling Settings

Definition at line 54 of file messages.h.

7.1.2.29 const uint8_t USU::SEL0 = 4

Bit offset for channel selection

Definition at line 41 of file max127.h.

7.1.2.30 const uint8_t USU::SET_CONTINUOUS_MODE = 0xC4

Set Continuous Mode

Definition at line 32 of file messages.h.

7.1.2.31 const uint8_t USU::STATIONARY_TEST = 0xDA

Stationary Test

Definition at line 53 of file messages.h.

7.1.2.32 const uint8_t USU::STOP_CONTINUOUS = 0xFA

Stop Continuous Mode (no reply)

Definition at line 61 of file messages.h.

7.1.2.33 const uint8_t USU::TEMPERATURES = 0xD1

Temperatures

Definition at line 45 of file messages.h.

7.1.2.34 const uint8_t USU::TIMER = 0xD7

Timer

Definition at line 51 of file messages.h.

7.1.2.35 const uint8_t USU::TRANSFER_TO_NONVOL_MEM = 0xD0

Transfer Quantity to Non-Volatile Memory

Definition at line 44 of file messages.h.

7.1.2.36 const uint8_t USU::WRITE_ACC_BIAS_CORRECTION = 0xC9

Write Accel Bias Correction

Definition at line 37 of file messages.h.

7.1.2.37 const uint8_t USU::WRITE_GYRO_BIAS_CORRECTION = 0xCA

Write Gyro Bias Correction

Definition at line 38 of file messages.h.

7.1.2.38 const uint8_t USU::WRITE_WORD_EEPROM = 0xE4

Write Word to EEPROM

Definition at line 57 of file messages.h.

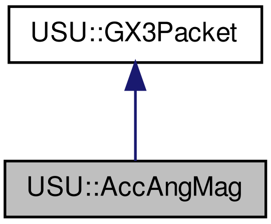
8 Class Documentation

8.1 USU::AccAngMag Class Reference

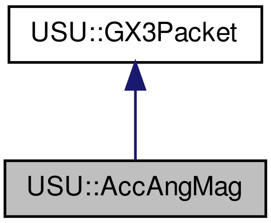
Representation for receiving acceleration, angular rate and magnetometer packets.

```
#include <messages.h>
```

Inheritance diagram for USU::AccAngMag:



Collaboration diagram for USU::AccAngMag:



Public Types

- enum { size = 43 }

Public Member Functions

- AccAngMag ()

  Creates an empty packet object.
- bool readFromSerial (SerialPort &serialPort)

  Read the information for the structure from the SerialPort.
- virtual void print (std::ostream &os) const

  Print the stored information to ostream object.

Public Attributes

- vector acc
- vector gyro
- vector mag
- unsigned int timer

8.1.1 Detailed Description

Representation for receiving acceleration, angular rate and magnetometer packets.

This class can be used with the commands which return 3 Vectors. The units are:

- acceleration: g
- angular rate: rad/s
- magnetic field: gauß

Definition at line 252 of file messages.h.

8.1.2 Member Enumeration Documentation

8.1.2.1 anonymous enum

Enumerator:

size

Definition at line 311 of file messages.h.

8.1.3 Constructor & Destructor Documentation

8.1.3.1 USU::AccAngMag::AccAngMag ( ) `[inline]`

Creates an empty packet object.

Definition at line 258 of file messages.h.

8.1.4 Member Function Documentation

8.1.4.1 virtual void USU::AccAngMag::print (std::ostream & *os*) const `[inline, virtual]`

Print the stored information to ostream object.

Format: timestamp,accX,accY,accZ,magX,magY,magZ,gyroX,gyroY,gyroZ

Parameters

os	

Implements USU::GX3Packet.

Definition at line 298 of file messages.h.

8.1.4.2 bool USU::AccAngMag::readFromSerial (SerialPort & *serialPort*) `[inline, virtual]`

Read the information for the structure from the SerialPort.

Parameters

serialPort	serialPort object from libserial

Returns

bool true if reading (and checksum) was successful, false otherwise

Implements USU::GX3Packet.

Definition at line 260 of file messages.h.

8.1.5 Member Data Documentation

8.1.5.1 vector USU::AccAngMag::acc

Vector containing the accelerometer data

Definition at line 305 of file messages.h.

8.1.5.2 vector USU::AccAngMag::gyro

Vector containing the gyroscope (angular rate) data

Definition at line 306 of file messages.h.

8.1.5.3 vector USU::AccAngMag::mag

Vector containing the magnetometer data

Definition at line 307 of file messages.h.

8.1.5.4 unsigned int USU::AccAngMag::timer

The value of the timestamp for the package

Definition at line 309 of file messages.h.

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

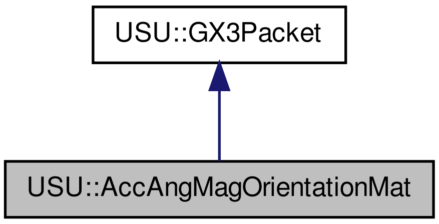
- include/messages.h

8.2 USU::AccAngMagOrientationMat Class Reference

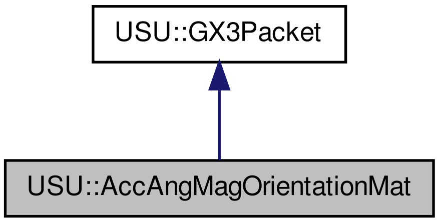
Representation for packets containing the 3 sensor vectors and orientation matrix This class can be used with the commands which return 3 Vectors and a 3x3 Matrix. The units are:

`#include <messages.h>`

Inheritance diagram for USU::AccAngMagOrientationMat:



Collaboration diagram for USU::AccAngMagOrientationMat:



Public Types

- enum { size = 79 }

Public Member Functions

- AccAngMagOrientationMat ()

Creates an empty packet object.

- bool readFromSerial (SerialPort &serialPort)

  Read the information for the structure from the SerialPort.
- virtual void print (std::ostream &os) const

  Print the stored information to ostream object.

Public Attributes

- vector acc
- vector gyro
- vector mag
- matrix orientation
- unsigned int timer

8.2.1 Detailed Description

Representation for packets containing the 3 sensor vectors and orientation matrix This class can be used with the commands which return 3 Vectors and a 3x3 Matrix. The units are:

- acceleration: g
- angular rate: rad/s
- magnetic field: gauß

Definition at line 379 of file messages.h.

8.2.2 Member Enumeration Documentation

8.2.2.1 anonymous enum

Enumerator:

size

Definition at line 433 of file messages.h.

8.2.3 Constructor & Destructor Documentation

8.2.3.1 USU::AccAngMagOrientationMat::AccAngMagOrientationMat ( ) `[inline]`

Creates an empty packet object.

Definition at line 385 of file messages.h.

8.2.4 Member Function Documentation

8.2.4.1 virtual void **USU::AccAngMagOrientationMat::print** (std::ostream & *os*) const `[inline, virtual]`

Print the stored information to ostream object.

Format: timestamp,accX,accY,accZ,magX,magY,magZ,gyroX,gyroY,gyroZ,mat(0,[0..2]),mat(1,[0..2]),mat(2,[0..2])

Parameters

os	

Implements USU::GX3Packet.

Definition at line 415 of file messages.h.

8.2.4.2 bool **USU::AccAngMagOrientationMat::readFromSerial** (SerialPort & *serialPort*) `[inline, virtual]`

Read the information for the structure from the SerialPort.

Parameters

serialPort	serialPort object from libserial

Returns

bool true if reading (and checksum) was successful, false otherwise

Implements USU::GX3Packet.

Definition at line 387 of file messages.h.

8.2.5 Member Data Documentation

8.2.5.1 vector USU::AccAngMagOrientationMat::acc

Vector containing the accelerometer data

Definition at line 426 of file messages.h.

8.2.5.2 vector USU::AccAngMagOrientationMat::gyro

Vector containing the gyroscope (angular rate) data

Definition at line 427 of file messages.h.

8.2.5.3 vector USU::AccAngMagOrientationMat::mag

Vector containing the magnetometer data

Definition at line 428 of file messages.h.

8.2.5.4 matrix USU::AccAngMagOrientationMat::orientation

3x3 Matrix containing the orientation

Definition at line 430 of file messages.h.

8.2.5.5 unsigned int USU::AccAngMagOrientationMat::timer

The value of the timestamp for the package

Definition at line 431 of file messages.h.

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

- include/messages.h

8.3 Beagle_GPIO Class Reference

Wrapper class to access the GPIOs of the BeagleBone.

```
#include <Beagle_GPIO.h>
```

Public Types

- enum Beagle_GPIO_Status { kFail = 0, kSuccess = 1 }
- enum { kREVISION = 0x0, kSYSCONFIG = 0x10, kIRQSTATUS_RAW_0 = 0x24, kIRQSTATUS_RAW_1 = 0x28, kIRQSTATUS_0 = 0x2C, kIRQSTATUS_1 = 0x30, kIRQSTATUS_SET_0 = 0x34, kIRQSTATUS_SET_1 = 0x38, kIRQSTATUS_CLR_0 = 0x3C, kIRQSTATUS_CLR_1 = 0x40, kIRQWAKEN_0 = 0x44, kIRQWAKEN_1 = 0x48, kSYSSTATUS = 0x114, kCTRL = 0x130, kOE = 0x134, kDATAIN = 0x138, kDATAOUT = 0x13C, kLEVELDETECT0 = 0x140, kLEVELDETECT1 = 0x144, kRISINGDETECT = 0x148, kFALLINGDETECT = 0x14C, kDEBOUNCEENABLE = 0x150, kDEBOUNCINGTIME = 0x154, kCLEARDATAOUT = 0x190, kSETDATAOUT = 0x194 }
- enum Beagle_GPIO_Direction { kINPUT = 0, kOUTPUT = 1 }
- enum Pins { P8_1, P8_2, P8_3, P8_4, P8_5, P8_6, P8_7, P8_8, P8_9, P8_10, P8_11, P8_12, P8_13, P8_14, P8_15, P8_16, P8_17, P8_18, P8_19, P8_20, P8_21, P8_22, P8_23, P8_24, P8_25, P8_26, P8_27, P8_28, P8_29, P8_30, P8_31, P8_32, P8_33, P8_34, P8_35, P8_36, P8_37, P8_38, P8_39, P8_40, P8_41, P8_42, P8_43, P8_44, P8_45, P8_46, P9_1, P9_2, P9_3, P9_4, P9_5, P9_6, P9_7, P9_8, P9_9, P9_10, P9_11, P9_12, P9_13, P9_14, P9_15, P9_16, P9_17, P9_18, P9_19, P9_20, P9_21, P9_22, P9_23, P9_24, P9_25, P9_26, P9_27, P9_28, P9_29, P9_30, P9_31, P9_32, P9_33, P9_34, P9_35, P9_36, P9_37, P9_38, P9_39, P9_40, P9_41, P9_42, P9_43, P9_44, P9_45, P9_46 }

Public Member Functions

- Beagle_GPIO ()
- ∼Beagle_GPIO ()

- Beagle_GPIO_Status configurePin (unsigned short _pin, Beagle_GPIO_-Direction _direction)
- Beagle_GPIO_Status enablePinInterrupts (unsigned short _pin, bool _enable)
- Beagle_GPIO_Status writePin (unsigned short _pin, unsigned char _value)
- unsigned char readPin (unsigned short _pin)
- void openSPI (unsigned char _mode=0, unsigned char _bits=8, unsigned long _speed=4800000, unsigned short _delay=0)
- void closeSPI ()
- void sendSPIBuffer (unsigned long buffer, int size)
- bool isActive ()

Public Attributes

- enum Beagle_GPIO:: { ... } Beagle_GPIO_Registers
- enum Beagle_GPIO::Pins GPIO_Pins

Static Public Attributes

- static const int GPIO_Pin_Bank [ ]
- static const int GPIO_Pin_Id [ ]
- static const unsigned long GPIO_Pad_Control [ ]
- static const unsigned long GPIO_Control_Module_Registers = 0x44E10000
- static const unsigned long GPIO_Base [ ]

8.3.1 Detailed Description

Wrapper class to access the GPIOs of the BeagleBone.

Definition at line 54 of file Beagle_GPIO.h.

8.3.2 Member Enumeration Documentation

8.3.2.1 anonymous enum

Enumerator:

kREVISION

kSYSCONFIG

kIRQSTATUS_RAW_0

kIRQSTATUS_RAW_1

kIRQSTATUS_0

kIRQSTATUS_1

kIRQSTATUS_SET_0

kIRQSTATUS_SET_1

kIRQSTATUS_CLR_0

kIRQSTATUS_CLR_1

kIRQWAKEN_0

kIRQWAKEN_1

kSYSSTATUS

kCTRL

kOE

kDATAIN

kDATAOUT

kLEVELDETECT0

kLEVELDETECT1

kRISINGDETECT

kFALLINGDETECT

kDEBOUNCEENABLE

kDEBOUNCINGTIME

kCLEARDATAOUT

kSETDATAOUT

Definition at line 65 of file Beagle_GPIO.h.

8.3.2.2 enum Beagle_GPIO::Beagle_GPIO_Direction

Enumerator:

kINPUT

kOUTPUT

Definition at line 95 of file Beagle_GPIO.h.

8.3.2.3 enum Beagle_GPIO::Beagle_GPIO_Status

Enumerator:

kFail

kSuccess

Definition at line 58 of file Beagle_GPIO.h.

8.3.2.4 enum Beagle_GPIO::Pins

Enumerator:

P8_1

P8_2

P8_3

P8_4

P8_5

P8_6

P8_7

P8_8

P8_9

P8_10

P8_11

P8_12

P8_13

P8_14

P8_15

P8_16

P8_17

P8_18

P8_19

P8_20

P8_21

P8_22

P8_23

P8_24

P8_25

P8_26

P8_27

P8_28

P8_29

P8_30

P8_31

P8_32

P8_33

P8_34

P8_35

P8_36

P8_37

P8_38

P8_39

P8_40

P8_41

P8_42

P8_43

P8_44

P8_45

P8_46

P9_1

P9_2

P9_3

P9_4

P9_5

P9_6

P9_7

P9_8

P9_9

P9_10

P9_11

P9_12

P9_13

P9_14

P9_15

P9_16

P9_17

P9_18

P9_19

P9_20

P9_21

P9_22

P9_23

P9_24

P9_25

P9_26

P9_27

P9_28

P9_29

P9_30

P9_31

P9_32

P9_33

P9_34

P9_35

P9_36

P9_37

P9_38

P9_39

P9_40

P9_41

P9_42

P9_43

P9_44

P9_45

P9_46

Definition at line 102 of file Beagle_GPIO.h.

8.3.3 Constructor & Destructor Documentation

8.3.3.1 Beagle_GPIO::Beagle_GPIO ( )

Definition at line 127 of file Beagle_GPIO.cpp.

8.3.3.2 Beagle_GPIO::∼Beagle_GPIO ( )

Definition at line 172 of file Beagle_GPIO.cpp.

8.3.4 Member Function Documentation

8.3.4.1 void Beagle_GPIO::closeSPI ( )

Definition at line 363 of file Beagle_GPIO.cpp.

8.3.4.2 Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::configurePin (unsigned short *_pin*, Beagle_GPIO_Direction *_direction*)

Definition at line 183 of file Beagle_GPIO.cpp.

8.3.4.3 Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::enablePinInterrupts (unsigned short *_pin*, bool *_enable*)

Definition at line 216 of file Beagle_GPIO.cpp.

8.3.4.4 bool Beagle_GPIO::isActive ( ) `[inline]`

Definition at line 171 of file Beagle_GPIO.h.

8.3.4.5 void **Beagle_GPIO::openSPI** (unsigned char *_mode* = 0, unsigned char *_bits* = 8, unsigned long *_speed* = 4800000, unsigned short *_delay* = 0)

Definition at line 284 of file Beagle_GPIO.cpp.

8.3.4.6 unsigned char **Beagle_GPIO::readPin** (unsigned short *_pin*)

Definition at line 268 of file Beagle_GPIO.cpp.

8.3.4.7 void **Beagle_GPIO::sendSPIBuffer** (unsigned long *buffer*, int *size*)

Definition at line 377 of file Beagle_GPIO.cpp.

8.3.4.8 **Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::writePin** (unsigned short *_pin*, unsigned char *_value*)

Definition at line 248 of file Beagle_GPIO.cpp.

8.3.5 Member Data Documentation

8.3.5.1 enum { ... } **Beagle_GPIO::Beagle_GPIO_Registers**

8.3.5.2 const unsigned long **Beagle_GPIO::GPIO_Base** [static]

Initial value:

```
{
                0x44E07000,
                0x4804C000,
                0x481AC000,
                0x481AE000
}
```

Definition at line 139 of file Beagle_GPIO.h.

8.3.5.3 const unsigned long **Beagle_GPIO::GPIO_Control_Module_Registers = 0x44E10000** [static]

Definition at line 136 of file Beagle_GPIO.h.

8.3.5.4 const unsigned long **Beagle_GPIO::GPIO_Pad_Control** [static]

Initial value:

```
{
                0x0000, 0x0000, 0x0818, 0x081C, 0x0808,
                0x080C, 0x0890, 0x0894, 0x089C, 0x0898,
                0x0834, 0x0830, 0x0824, 0x0828, 0x083C,
                0x0838, 0x082C, 0x088C, 0x0820, 0x0884,
                0x0880, 0x0814, 0x0810, 0x0804, 0x0800,
                0x087C, 0x08E0, 0x08E8, 0x08E4, 0x08EC,
                0x08D8, 0x08DC, 0x08D4, 0x08CC, 0x08D0,
                0x08C8, 0x08C0, 0x08C4, 0x08B8, 0x08BC,
                0x08B0, 0x08B4, 0x08A8, 0x08AC, 0x08A0,
               0x08A4,
               0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
               0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
               0x0870, 0x0878, 0x0874, 0x0848, 0x0840,
               0x084C, 0x095C, 0x0958, 0x097C, 0x0978,
               0x0954, 0x0950, 0x0844, 0x0984, 0x09AC,
               0x0980, 0x09A4, 0x099C, 0x0994, 0x0998,
               0x0990, 0x0000, 0x0000, 0x0000, 0x0000,
               0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
               0x09B4, 0x0964, 0x0000, 0x0000, 0x0000,
               0x0000
}
```

Definition at line 133 of file Beagle_GPIO.h.

8.3.5.5 const int Beagle_GPIO::GPIO_Pin_Bank [static]

Initial value:

```
{
               -1, -1,  1,  1,  1,
                1,  2,  2,  2,  2,
                1,  1,  0,  0,  1,
                1,  0,  2,  0,  1,
                1,  1,  1,  1,  1,
                1,  2,  2,  2,  2,
                0,  0,  0,  2,  0,
                2,  2,  2,  2,  2,
                2,  2,  2,  2,  2,
                2,
               -1, -1, -1, -1, -1,
               -1, -1, -1, -1, -1,
                0,  1,  0,  1,  1,
                1,  0,  0,  0,  0,
                0,  0,  1,  0,  3,
                0,  3,  3,  3,  3,
                3, -1, -1, -1, -1,
               -1, -1, -1, -1, -1,
                0,  0, -1, -1, -1,
               -1
}
```

Definition at line 127 of file Beagle_GPIO.h.

8.3.5.6 const int Beagle_GPIO::GPIO_Pin_Id [static]

Initial value:

```
{
               -1, -1,  6,  7,  2,
                3,  2,  3,  5,  4,
               13, 12, 23, 26, 15,
               14, 27,  1, 22, 31,
               30,  5,  4,  1,  0,
               29, 22, 24, 23, 25,
               10, 11,  9, 17,  8,
               16, 14, 15, 12, 13,
               10, 11,  8,  9,  6,
                7,
               -1, -1, -1, -1, -1,
               -1, -1, -1, -1, -1,
               30, 28, 31, 18, 16,
               19,  5,  4, 13, 12,
                3,  2, 17, 15, 21,
               14, 19, 17, 15, 16,
                14, -1, -1, -1, -1,
                -1, -1, -1, -1, -1,
                20,  7, -1, -1, -1,
                -1
}
```

Definition at line 130 of file Beagle_GPIO.h.

8.3.5.7 enum Beagle_GPIO::Pins Beagle_GPIO::GPIO_Pins

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

- include/Beagle_GPIO.h
- src/Beagle_GPIO.cpp

8.4 cPWM Class Reference

Wrapper class to access the PWM-devices of the BeagleBone.

`#include <cPWM.h>`

Public Types

- enum Polarity { ActiveHigh, ActiveLow }

Public Member Functions

- cPWM (int id)

  Simple C++ class wrapper for beaglebone PWM eHRPWM interface.
- virtual ∼cPWM ()
- void DutyA_ns (unsigned int nanoseconds)
- void DutyA_percent (unsigned int percent)
- void DutyB_ns (unsigned int nanoseconds)
- void DutyB_percent (unsigned int percent)
- void Period_ns (unsigned int nanoseconds)
- void Period_freq (unsigned int freq_Hz)
- void PolarityA (cPWM::Polarity polarity)
- void RunA ()
- void StopA ()
- void PolarityB (cPWM::Polarity polarity)
- void RunB ()
- void StopB ()

8.4.1 Detailed Description

Wrapper class to access the PWM-devices of the BeagleBone.

Definition at line 24 of file cPWM.h.

8.4.2 Member Enumeration Documentation

8.4.2.1 enum cPWM::Polarity

Enumerator:

- ***ActiveHigh***
- ***ActiveLow***

Definition at line 27 of file cPWM.h.

8.4.3 Constructor & Destructor Documentation

8.4.3.1 cPWM::cPWM (int *id*)

Simple C++ class wrapper for beaglebone PWM eHRPWM interface.

This class wraps the PWMss of the beaglebone, but it accesses the PWMss by means of the sysfs interface, so probably other systems are supported as well. The sysfs filenames are defined in cPWM.h. The constructor just opens the sysfs files but doesn't write anything, so in order to properly use the PWMss you need to follow all the steps (frequency, period, polarity) before calling run.

Parameters

in	*id*	id of the PWMss to be initializaed. There are 3 of them, eHRPWM0 thru 2.

Returns

a cPWM object

TODO: Add clock selection (mmap). By now you must use setPWMReg.py method FIXME: pin mux settings should be done here? or at a highet level?

Definition at line 33 of file cPWM.cpp.

8.4.3.2 cPWM::∼cPWM ( ) `[virtual]`

cPWM Destructor, stops the PWMss

Definition at line 261 of file cPWM.cpp.

8.4.4 Member Function Documentation

8.4.4.1 void cPWM::DutyA_ns (unsigned int *nanoseconds*)

Set the duty cycle for A channel of the PWMss

Parameters

in	*nanoseconds,-:*	duty cycle time in nanoseconds for A channel

Definition at line 98 of file cPWM.cpp.

8.4.4.2 void cPWM::DutyA_percent (unsigned int *percent*)

Set the duty cycle for A channel of the PWMss

Parameters

in	*percent,:*	duty cycle time in percent for A channel

Definition at line 113 of file cPWM.cpp.

8.4.4.3 void cPWM::DutyB_ns (unsigned int *nanoseconds*)

Set the duty cycle for B channel of the PWMss

Parameters

in	*nanoseconds,-:*	duty cycle time in nanoseconds for B channel

Definition at line 127 of file cPWM.cpp.

8.4.4.4 void cPWM::DutyB_percent (unsigned int *percent*)

Set the duty cycle for B channel of the PWMss

Parameters

in	*percent,:*	duty cycle time in percent for B channel

Definition at line 143 of file cPWM.cpp.

8.4.4.5 void cPWM::Period_freq (unsigned int *freq_Hz*)

Set the period for the PWMss

Parameters

in	*freq_Hz,:*	PWM frequency in Hz

Definition at line 171 of file cPWM.cpp.

8.4.4.6 void cPWM::Period_ns (unsigned int *nanoseconds*)

Set the period for the PWMss

Parameters

in	*nanoseconds,-:*	period time in nanoseconds

Definition at line 158 of file cPWM.cpp.

8.4.4.7 void cPWM::PolarityA (cPWM::Polarity *polarity*)

Set the polarity for the A channel of the PWMss

Parameters

in	*polarity*	polarity

Definition at line 184 of file cPWM.cpp.

8.4.4.8 void cPWM::PolarityB (cPWM::Polarity *polarity*)

Set the polarity for the B channel of the PWMss

Parameters

in	*polarity*	polarity

Definition at line 224 of file cPWM.cpp.

8.4.4.9 void cPWM::RunA ( )

Set the A channel to run status

Definition at line 201 of file cPWM.cpp.

8.4.4.10 void cPWM::RunB ( )

Set the B channel to run

Definition at line 241 of file cPWM.cpp.

8.4.4.11 void cPWM::StopA ( )

Stop the A channel

Definition at line 212 of file cPWM.cpp.

8.4.4.12 void cPWM::StopB ( )

Stop the B channel

Definition at line 251 of file cPWM.cpp.

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

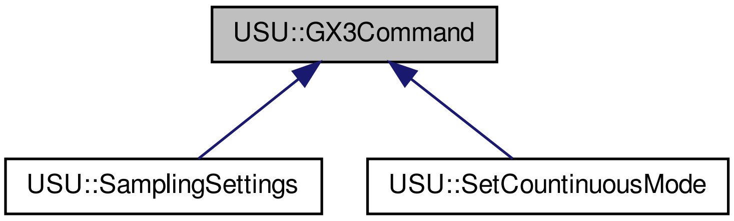
- include/cPWM.h
- src/cPWM.cpp

8.5 USU::GX3Command Class Reference

Base class for commands send to the 3DM-GX3-25.

`#include <messages.h>`

Inheritance diagram for USU::GX3Command:



Public Member Functions

- virtual bool sendCommand (SerialPort &serialPort)=0
- virtual bool checkResponse (uint8_t ∗buffer)=0

8.5.1 Detailed Description

Base class for commands send to the 3DM-GX3-25.

Just an empty base class, so that all commands share the same base class.

TODO: Implement sendCommand in base class instead of in each class separately?

Definition at line 445 of file messages.h.

8.5.2 Member Function Documentation

8.5.2.1 virtual bool USU::GX3Command::checkResponse (uint8_t * *buffer*) [pure virtual]

Implemented in USU::SamplingSettings, and USU::SetCountinuousMode.

8.5.2.2 virtual bool USU::GX3Command::sendCommand (SerialPort & *serialPort*) [pure virtual]

Implemented in USU::SamplingSettings, and USU::SetCountinuousMode.

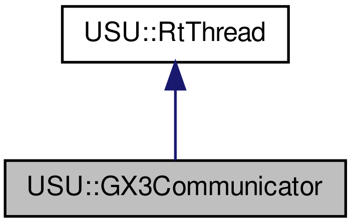
The documentation for this class was generated from the following file:

- include/messages.h

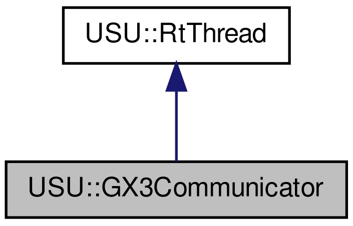
8.6 USU::GX3Communicator Class Reference

`#include <gx3communicator.h>`

Inheritance diagram for USU::GX3Communicator:



Collaboration diagram for USU::GX3Communicator:



Public Member Functions

- GX3Communicator (int priority, const char ∗serialDevice, SerialPort::BaudRate baudRate=SerialPort::BAUD_115200)

  Constructor of the class.
- void initialize ()

  Initialize the SerialPort and the MicroStrain IMU.
- virtual void run ()

  Thread routine.
- void stop ()

  Signals the thread to stop.
- void pop ()

  Delete the first element of the FIFO.
- bool isEmpty ()

  Check if the FIFO is empty.
- unsigned size ()

  Return the number of elements in the FIFO.
- packet_ptr & front ()

  Return the first element from the FIFO.

8.6.1 Detailed Description

Definition at line 46 of file gx3communicator.h.

8.6.2 Constructor & Destructor Documentation

8.6.2.1 GX3Communicator::GX3Communicator (int *priority*, const char ∗ *serialDevice*, SerialPort::BaudRate *baudRate* = `SerialPort::BAUD_115200`)

Constructor of the class.

Sets up the serial port and thread attributes.

Parameters

priority	Priority of the pthread (1..99)
serialDevice	Name of the serial device
baudRate	Baud rate for the serial device (if different from 115200)

Definition at line 47 of file gx3communicator.cpp.

8.6.3 Member Function Documentation

8.6.3.1 packet_ptr& USU::GX3Communicator::front ( ) `[inline]`

Return the first element from the FIFO.

TODO: Make a blocking version of it

Returns

AccAngMag the first element

Definition at line 111 of file gx3communicator.h.

8.6.3.2 void GX3Communicator::initialize ( )

Initialize the SerialPort and the MicroStrain IMU.

Definition at line 53 of file gx3communicator.cpp.

8.6.3.3 bool USU::GX3Communicator::isEmpty ( ) `[inline]`

Check if the FIFO is empty.

Returns

bool true, if empty

Definition at line 95 of file gx3communicator.h.

8.6.3.4 void USU::GX3Communicator::pop ( ) `[inline]`

Delete the first element of the FIFO.

Definition at line 87 of file gx3communicator.h.

8.6.3.5 void GX3Communicator::run ( ) [virtual]

Thread routine.

- Set sampling settings of 3DM
- Start continuous mode
- Poll serial port for newly arrived packages
- Convert binary data
- TODO: Send new package to KalmanFilter

TODO: Error

TODO: Error?

Implements USU::RtThread.

Definition at line 73 of file gx3communicator.cpp.

8.6.3.6 unsigned USU::GX3Communicator::size ( ) [inline]

Return the number of elements in the FIFO.

Returns

unsigned number of elements

Definition at line 102 of file gx3communicator.h.

8.6.3.7 void USU::GX3Communicator::stop ( ) [inline]

Signals the thread to stop.

Definition at line 82 of file gx3communicator.h.

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

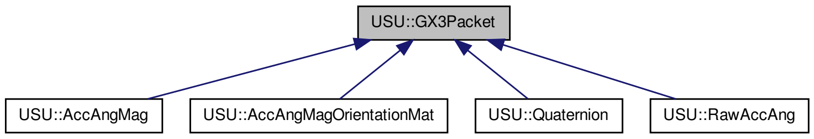
- include/gx3communicator.h
- src/gx3communicator.cpp

8.7 USU::GX3Packet Class Reference

Abstract base class for received packets.

```
#include <messages.h>
```

Inheritance diagram for USU::GX3Packet:



Public Member Functions

- virtual bool readFromSerial (SerialPort &serialPort)=0

    Read the information for the structure from the SerialPort.
- virtual void print (std::ostream &os) const =0

    Print the information of the GX3Packet to an ostream object.

Static Public Member Functions

- static bool calculateChecksum (uint8_t ∗buffer, unsigned int length)

    Calculates the checksum of a received byte array.

Static Protected Member Functions

- static vector createVector (uint8_t ∗buffer)

    Creates a Eigen::Vector3f consisting of 3 floats from 12 sucessive bytes.
- static unsigned int createUInt (uint8_t ∗buffer)

    Creates an unsigned integer from 4 successive bytes.
- static void createMatrix (uint8_t ∗buffer, matrix &mat)

    Creates a Eigen::Matrix3f from byte array.

8.7.1 Detailed Description

Abstract base class for received packets.

The class provides some useful function available to all derived classes such as checksum calculation and creation of vectors and matrizes from the received binary data.

Definition at line 79 of file messages.h.

8.7.2 Member Function Documentation

8.7.2.1 static bool **USU::GX3Packet::calculateChecksum** (uint8_t * *buffer*, unsigned int *length*) `[inline, static]`

Calculates the checksum of a received byte array.

Parameters

buffer	pointer to the byte array
length	length of the byte array

Returns

bool true: checksum matches, false: checksum does not match

Definition at line 107 of file messages.h.

8.7.2.2 static void **USU::GX3Packet::createMatrix** (uint8_t * *buffer*, **matrix** & *mat*) `[inline, static, protected]`

Creates a Eigen::Matrix3f from byte array.

NOTE: Make sure that the endianess of the host system and the 3DM match. The endianess of the sent floats can be set with the SamplingSettings command.

Parameters

buffer	Pointer to the byte array
mat	reference to a matrix which will be filled with the data from the byte array

Definition at line 156 of file messages.h.

8.7.2.3 static unsigned int **USU::GX3Packet::createUInt** (uint8_t * *buffer*) `[inline, static, protected]`

Creates an unsigned integer from 4 successive bytes.

Parameters

buffer	Pointer to the byte array

Returns

unsigned int created unsigned integer

Definition at line 142 of file messages.h.

8.7.2.4 static vector **USU::GX3Packet::createVector** (uint8_t * *buffer*) `[inline, static, protected]`

Creates a Eigen::Vector3f consisting of 3 floats from 12 sucessive bytes.

NOTE: Make sure that the endianess of the host system and the 3DM match. The endianess of the sent floats can be set with the SamplingSettings command.

Parameters

buffer	Pointer to the byte array

Returns

vector vector created from the byte array

Definition at line 129 of file messages.h.

8.7.2.5 virtual void **USU::GX3Packet::print** (std::ostream & *os*) const `[pure virtual]`

Print the information of the GX3Packet to an ostream object.

Enables convenient data recording of all different GX3Packet classes. Uses csv format; every packet is a single line (without std::endl).

Parameters

os	

Implemented in USU::AccAngMagOrientationMat, USU::Quaternion, USU::AccAngMag, and USU::RawAccAng.

8.7.2.6 virtual bool **USU::GX3Packet::readFromSerial** (SerialPort & *serialPort*) `[pure virtual]`

Read the information for the structure from the SerialPort.

Parameters

serialPort	serialPort object from libserial

Returns

bool true if reading (and checksum) was successful, false otherwise

Implemented in USU::AccAngMagOrientationMat, USU::Quaternion, USU::AccAngMag, and USU::RawAccAng.

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

- include/messages.h

8.8 I2CBus Class Reference

Wrapper class for I2C-bus communication.

```
#include <I2CBus.h>
```

Public Member Functions

- I2CBus (const char ∗deviceName)

  Constructor.
- ∼I2CBus ()

  Destructor.
- void addressSet (uint8_t address)

  Set the address of the I2C device the bus will read and write data to.
- void writeByte (uint8_t command, uint8_t data)

  Write a byte to the register command.
- void writeByte (uint8_t data)

  Write a byte without a specifying a register.
- uint8_t readByte (uint8_t command)

  Read a byte from the register command.
- uint8_t readByte ()

  Read a byte directly without specifying a register.
- uint16_t readWord (uint8_t command)

  Read a word (2 bytes) from the register command.
- uint16_t readWord ()

  Read a word (2 bytes) directly without specifying a register.
- int tryReadByte (uint8_t command)

  Tries to read a byte from register command.
- void readBlock (uint8_t command, uint8_t size, uint8_t ∗data)

  Read a block of data from the device starting at register command.

8.8.1 Detailed Description

Wrapper class for I2C-bus communication.

Definition at line 16 of file I2CBus.h.

8.8.2 Constructor & Destructor Documentation

8.8.2.1 I2CBus::I2CBus (const char ∗ *deviceName*)

Constructor.

Sets up the interface to the I2C-bus deviceName

Parameters

deviceName	Name of the I2C-bus device

Definition at line 8 of file I2CBus.cpp.

8.8.2.2 I2CBus::∼I2CBus ( )

Destructor.

Definition at line 17 of file I2CBus.cpp.

8.8.3 Member Function Documentation

8.8.3.1 void I2CBus::addressSet (uint8_t *address*)

Set the address of the I2C device the bus will read and write data to.

Parameters

address	7-bit address (trailing 0)

Definition at line 22 of file I2CBus.cpp.

8.8.3.2 void I2CBus::readBlock (uint8_t *command,* uint8_t *size,* uint8_t ∗ *data*)

Read a block of data from the device starting at register command.

Parameters

command	Register to start reading from
size	Number of bytes to read
data	Allocated buffer with length of at least size

Definition at line 98 of file I2CBus.cpp.

8.8.3.3 uint8_t I2CBus::readByte (uint8_t *command*)

Read a byte from the register command.

Parameters

command	Register to read from

Returns

uint8_t Value of the register command

Definition at line 49 of file I2CBus.cpp.

8.8.3.4 uint8_t I2CBus::readByte ( )

Read a byte directly without specifying a register.

Read a byte directly from the device set with addressSet() without specifying a register.

Returns

uint8_t Value of the read data byte

Definition at line 61 of file I2CBus.cpp.

8.8.3.5 uint16_t **I2CBus::readWord** (uint8_t *command*)

Read a word (2 bytes) from the register command.

Parameters

command	Register to read the word from

Returns

uint16_t Value of the register command

Definition at line 71 of file I2CBus.cpp.

8.8.3.6 uint16_t **I2CBus::readWord** ( )

Read a word (2 bytes) directly without specifying a register.

Read a word (2 bytes) directly from the device set with addressSet() without specifying a register

Returns

uint16_t Value of the read data word

Definition at line 81 of file I2CBus.cpp.

8.8.3.7 int **I2CBus::tryReadByte** (uint8_t *command*)

Tries to read a byte from register command.

Difference to readByte(uint8_t) is, that this function won't check if the reading was successful. Returns the value of the register if successful and -1 if the read failed.

Parameters

command	

Returns

int

Definition at line 92 of file I2CBus.cpp.

8.8.3.8 void **I2CBus::writeByte** (uint8_t *command,* uint8_t *data*)

Write a byte to the register command.

Parameters

command	Register to write the byte to
data	Byte of data to write to the device set with addressSet()

Definition at line 31 of file I2CBus.cpp.

8.8.3.9 void **I2CBus::writeByte** (uint8_t *data*)

Write a byte without a specifying a register.

Parameters

data	Byte of data which will be written directly to the device set with address-Set()

Definition at line 40 of file I2CBus.cpp.

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

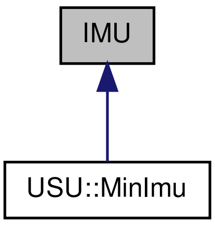
- include/I2CBus.h
- src/I2CBus.cpp

8.9 IMU Class Reference

Virtual base class for IMU.

`#include <IMU.h>`

Inheritance diagram for IMU:



Public Member Functions

- virtual vector readMag ()=0
- virtual vector readAcc ()=0

- virtual vector readGyro ()=0
- void read ()
- virtual void enable ()=0

Public Attributes

- int_vector raw_m
- int_vector raw_a
- int_vector raw_g

8.9.1 Detailed Description

Virtual base class for IMU.

Derive this class to make your own IMU-class.

Definition at line 13 of file IMU.h.

8.9.2 Member Function Documentation

8.9.2.1 virtual void **IMU::enable** ( ) [pure virtual]

Implemented in USU::MinImu.

8.9.2.2 void **IMU::read** ( ) [inline]

Definition at line 19 of file IMU.h.

8.9.2.3 virtual vector **IMU::readAcc** ( ) [pure virtual]

Implemented in USU::MinImu.

8.9.2.4 virtual vector **IMU::readGyro** ( ) [pure virtual]

Implemented in USU::MinImu.

8.9.2.5 virtual vector **IMU::readMag** ( ) [pure virtual]

Implemented in USU::MinImu.

8.9.3 Member Data Documentation

8.9.3.1 int_vector IMU::raw_a

Definition at line 29 of file IMU.h.

8.9.3.2 int_vector IMU::raw_g

Definition at line 29 of file IMU.h.

8.9.3.3 int_vector IMU::raw_m

Definition at line 29 of file IMU.h.

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

- include/IMU.h

8.10 L3G Class Reference

Class to manage the communication to the L3G gyroscope via the I2C-bus.

```
#include <L3G.h>
```

Public Member Functions

- L3G (const char ∗i2cDeviceName)
- void enable (void)

  Puts the chip into active sampling mode.
- void writeReg (uint8_t reg, uint8_t value)

  Write value to register reg.
- uint8_t readReg (uint8_t reg)

  Read the value from register reg.
- void read ()

  Reads the current raw angular rates into g.

Public Attributes

- int g [3]

8.10.1 Detailed Description

Class to manage the communication to the L3G gyroscope via the I2C-bus.

Definition at line 44 of file L3G.h.

8.10.2 Constructor & Destructor Documentation

8.10.2.1 L3G::L3G (const char ∗ *i2cDeviceName*)

Parameters

i2cDevice-Name	

Definition at line 9 of file L3G.cpp.

8.10.3 Member Function Documentation

8.10.3.1 void L3G::enable (void)

Puts the chip into active sampling mode.

Definition at line 28 of file L3G.cpp.

8.10.3.2 void L3G::read ( )

Reads the current raw angular rates into g.

Definition at line 46 of file L3G.cpp.

8.10.3.3 uint8_t L3G::readReg (uint8_t *reg*)

Read the value from register reg.

Parameters

reg	Register address to read from

Returns

uint8_t Value read from the register reg

Definition at line 41 of file L3G.cpp.

8.10.3.4 void L3G::writeReg (uint8_t *reg,* uint8_t *value*)

Write value to register reg.

TODO: Make registers enum, so that writing to wrong register impossible?

Parameters

reg	Register address to write to
value	Value to write to the register reg

Definition at line 36 of file L3G.cpp.

8.10.4 Member Data Documentation

8.10.4.1 int L3G::g[3]

Gyro raw angular velocity readings

Definition at line 54 of file L3G.h.

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

- include/L3G.h
- src/L3G.cpp

8.11 USU::Lock Class Reference

Wrapper class for pthread mutexes.

`#include <Lock.h>`

Public Member Functions

- Lock ()
- virtual ∼Lock ()
- void lock ()
- void unlock ()

8.11.1 Detailed Description

Wrapper class for pthread mutexes.

Definition at line 25 of file Lock.h.

8.11.2 Constructor & Destructor Documentation

8.11.2.1 USU::Lock::Lock ( ) `[inline]`

Constructor: Creates the pthread-mutex

Definition at line 45 of file Lock.h.

8.11.2.2 USU::Lock::∼Lock ( ) `[inline, virtual]`

Destructor: Frees the pthread-mutex

Definition at line 55 of file Lock.h.

8.11.3 Member Function Documentation

8.11.3.1 void USU::Lock::lock ( ) `[inline]`

Locks the mutex

Definition at line 66 of file Lock.h.

8.11.3.2 void USU::Lock::unlock ( ) `[inline]`

Unlocks the mutex

Definition at line 72 of file Lock.h.

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

- include/Lock.h

8.12 LSM303 Class Reference

Class to manage communication to the LSM303 compass via the I2C-bus.

`#include <LSM303.h>`

Public Member Functions

- LSM303 (const char ∗i2cDeviceName)

  Constructor.
- void enable (void)

  Puts both (accelerometer and magnetometer) into active sampling mode.
- void writeAccReg (uint8_t reg, uint8_t value)

  Write value to the accelerometer register reg.
- uint8_t readAccReg (uint8_t reg)

  Read the value from accelerometer register reg.
- void writeMagReg (uint8_t reg, uint8_t value)

  Write value to the magnetometer register reg.
- uint8_t readMagReg (uint8_t reg)

  Read the value from magnetometer register reg.
- void readAcc (void)

  Reads the current raw acceleration vector into a.
- void readMag (void)

  Reads the current raw magnetic field vector into m.
- void read (void)

  Read both (accelerometer and magnetometer) into a and m respectively.

Public Attributes

- int a [3]
- int m [3]

8.12.1 Detailed Description

Class to manage communication to the LSM303 compass via the I2C-bus.

LSM303 has a 3-axis accelerometer and a 3-axis magnetometer on a single chip and the same I2C-bus. This class manages the interface to both of them and handles the read out procedure for the analog values. Check the data sheet for more details of the settings.

Definition at line 88 of file LSM303.h.

8.12.2 Constructor & Destructor Documentation

8.12.2.1 LSM303::LSM303 (const char ∗ *i2cDeviceName*)

Constructor.

Sets up the accelerometer and magnetometer on the given I2C-bus.

Parameters

i2cDevice-Name	Device name of the I2C-bus

Definition at line 22 of file LSM303.cpp.

8.12.3 Member Function Documentation

8.12.3.1 void LSM303::enable (void)

Puts both (accelerometer and magnetometer) into active sampling mode.

Definition at line 49 of file LSM303.cpp.

8.12.3.2 void LSM303::read (void)

Read both (accelerometer and magnetometer) into a and m respectively.

Definition at line 119 of file LSM303.cpp.

8.12.3.3 void LSM303::readAcc (void)

Reads the current raw acceleration vector into a.

Definition at line 94 of file LSM303.cpp.

8.12.3.4 uint8_t LSM303::readAccReg (uint8_t *reg*)

Read the value from accelerometer register reg.

Parameters

reg	Register address to read from

Returns

uint8_t Value read from the register reg

Definition at line 32 of file LSM303.cpp.

8.12.3.5 void LSM303::readMag (void)

Reads the current raw magnetic field vector into m.

Definition at line 104 of file LSM303.cpp.

8.12.3.6 uint8_t LSM303::readMagReg (uint8_t *reg*)

Read the value from magnetometer register reg.

Parameters

reg	Register address to read from

Returns

uint8_t Value read from the register reg

Definition at line 27 of file LSM303.cpp.

8.12.3.7 void LSM303::writeAccReg (uint8_t *reg*, uint8_t *value*)

Write value to the accelerometer register reg.

Parameters

reg	Register address to write to
value	Value to write to the register reg

Definition at line 42 of file LSM303.cpp.

8.12.3.8 void LSM303::writeMagReg (uint8_t *reg*, uint8_t *value*)

Write value to the magnetometer register reg.

Parameters

reg	Register address to write to
value	Value to write to the register reg

Definition at line 37 of file LSM303.cpp.

8.12.4 Member Data Documentation

8.12.4.1 int LSM303::a[3]

Raw accelerometer readings

Definition at line 91 of file LSM303.h.

8.12.4.2 int LSM303::m[3]

Magnetometer readings

Definition at line 92 of file LSM303.h.

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

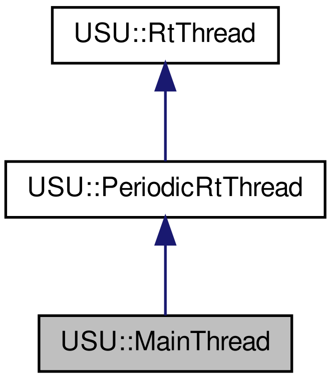
- include/LSM303.h
- src/LSM303.cpp

8.13 USU::MainThread Class Reference

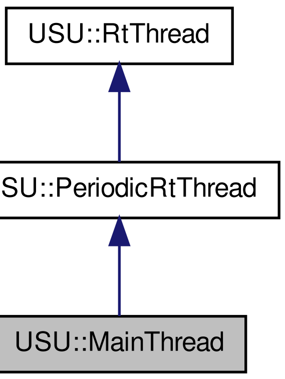
Represents the Periodic Thread class for state estimation.

```
#include <mainthread.h>
```

Inheritance diagram for USU::MainThread:



Collaboration diagram for USU::MainThread:



Classes

- struct **Command**

  Struct representing a single command point.

Public Types

- enum Mode { SimpleControl, CollectPololuData, CollectMicroStrainData, Collect-Data }

Public Member Functions

- MainThread (int priority, unsigned int period_us, const char ∗i2cImu, const char ∗i2cMotor)

  Constructor of the class.
- virtual void run ()

  Thread routine.
- void stop ()

  Signals the thread to stop.
- bool getState ()

  Returns the current system state estimate.
- void initializeModeSimpleControl (std::string trajFilename, float pgain)
- Mode getMode () const
- void setMode (const Mode &value)

8.13.1 Detailed Description

Represents the Periodic Thread class for state estimation.

This class is derived from PeriodicRtThread. It initializes the interface to the MinIMU9v2 and estimates the system state using Kalman filtering techniques. The state estimate can be accessed from other threads (protected by mutex).

TODO:

- Implement kalman filter for state estimate
- change name to something more meaningful?

Definition at line 38 of file mainthread.h.

8.13.2 Member Enumeration Documentation

8.13.2.1 enum USU::MainThread::Mode

Enumerator:

SimpleControl

CollectPololuData

CollectMicroStrainData

CollectData

Definition at line 41 of file mainthread.h.

8.13.3 Constructor & Destructor Documentation

8.13.3.1 MainThread::MainThread (int *priority*, unsigned int *period_us*, const char ∗ *i2cImu*, const char ∗ *i2cMotor*)

Constructor of the class.

Initializes the interface to the MinIMU9 sensors and to the 3DM-GX3. Sets up the motor controller.

Parameters

priority	priority of the underlying periodic thread
period_us	period (in us) of the underlying periodic thread
i2cImu	name of the I2C-device for the IMU (e.g. /dev/i2c-1)
i2cMotor	name of the I2C-device for the Motors (e.g. /dev/i2c-2)

Definition at line 49 of file mainthread.cpp.

8.13.4 Member Function Documentation

8.13.4.1 MainThread::Mode MainThread::getMode ( ) const

Definition at line 304 of file mainthread.cpp.

8.13.4.2 bool MainThread::getState ( )

Returns the current system state estimate.

Copies the current system state estimate. Acquires mutex before acessing the internal variable to avoid read/write-conflicts.

Returns

bool Current system state TODO: Currently only dummy variable. Replace with actual state representation (quaternion?) Probably not necessary anymore

Definition at line 72 of file mainthread.cpp.

8.13.4.3 void MainThread::initializeModeSimpleControl (std::string *trajFilename,* float *pgain*)

Definition at line 78 of file mainthread.cpp.

8.13.4.4 void MainThread::run ( ) `[virtual]`

Thread routine.

Current scenario is:

- Get quaternion data from MicroStrain at constant rate
- Hand this state estimate to the motor controller.

TODO: Develop scenario using Kalman-Filter

Implements USU::PeriodicRtThread.

Definition at line 55 of file mainthread.cpp.

8.13.4.5 void MainThread::setMode (const Mode & *value*)

Definition at line 309 of file mainthread.cpp.

8.13.4.6 void USU::MainThread::stop ( ) `[inline]`

Signals the thread to stop.

Definition at line 78 of file mainthread.h.

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

- include/mainthread.h
- src/mainthread.cpp

8.14 USU::Max127 Class Reference

Class representing the MAX127 ADC.

`#include <max127.h>`

Public Member Functions

- Max127 (const char ∗i2cdevice)

  Constructor.
- int16_t readRaw (uint8_t channel)

  Returns the raw integer measurement of the selected channel.
- float readVoltage (unsigned int channel)

  Returns the measurement of the selected channel in volts.

8.14.1 Detailed Description

Class representing the MAX127 ADC.

Provides simple functionality to read the channels. Uses the I2CBus class for communication.

Definition at line 52 of file max127.h.

8.14.2 Constructor & Destructor Documentation

8.14.2.1 Max127::Max127 (const char ∗ *i2cdevice*)

Constructor.

Initializes the I2C-connection

Parameters

i2cdevice	device name of the i2c-bus (e.g. /dev/i2c-1)

Definition at line 14 of file max127.cpp.

8.14.3 Member Function Documentation

8.14.3.1 int16_t Max127::readRaw (uint8_t *channel*)

Returns the raw integer measurement of the selected channel.

At the moment assumens bipolar operation. The range is [-2048, 2047]

Parameters

channel	channel to read

Returns

int16_t signed integer representing the measurement

Definition at line 20 of file max127.cpp.

8.14.3.2 float Max127::readVoltage (unsigned int *channel*)

Returns the measurement of the selected channel in volts.

At the moment assumes fullscale of 10 V (bipolar +-5V or unipolar)

Parameters

channel	channel to read

Returns

float measured voltage in V

Definition at line 35 of file max127.cpp.

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

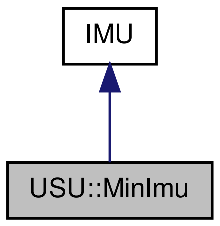
- include/max127.h
- src/max127.cpp

8.15 USU::MinImu Class Reference

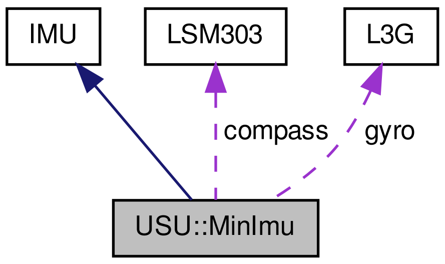
Class to manage the communication to the Pololu MinIMU9.

`#include <minimu.h>`

Inheritance diagram for USU::MinImu:



Collaboration diagram for USU::MinImu:



Public Member Functions

- MinImu (const char ∗i2cDeviceName)

  Constructor.
- virtual vector readMag ()

  Reads the magnetometer and return a vector of raw values.
- virtual vector readAcc ()

  Reads the accelerometer and return a vector with units in g.
- virtual vector readGyro ()

  Reads the gyroscope and returns a vector with units in degrees/s.
- virtual void enable ()

  Enables compass and gyroscope, i.e. starts the sampling on these devices.

Public Attributes

- LSM303 compass
- L3G gyro

8.15.1 Detailed Description

Class to manage the communication to the Pololu MinIMU9.

Definition at line 32 of file minimu.h.

8.15.2 Constructor & Destructor Documentation

8.15.2.1 MinImu::MinImu (const char ∗ *i2cDeviceName*)

Constructor.

Initializes the compass and gyroscope.

Parameters

i2cDevice-Name	Name of the I2C device the IMU is connected to

Definition at line 5 of file minimu.cpp.

8.15.3 Member Function Documentation

8.15.3.1 void MinImu::enable (void) `[virtual]`

Enables compass and gyroscope, i.e. starts the sampling on these devices.

Implements IMU.

Definition at line 11 of file minimu.cpp.

8.15.3.2 vector MinImu::readAcc (void) `[virtual]`

Reads the accelerometer and return a vector with units in g.

Returns

vector

Implements IMU.

Definition at line 28 of file minimu.cpp.

8.15.3.3 vector MinImu::readGyro ( ) `[virtual]`

Reads the gyroscope and returns a vector with units in degrees/s.

Returns

vector

Implements IMU.

Definition at line 17 of file minimu.cpp.

8.15.3.4 vector MinImu::readMag (void) `[virtual]`

Reads the magnetometer and return a vector of raw values.

TODO: Transform into gauss?

Returns

vector

Implements IMU.

Definition at line 39 of file minimu.cpp.

8.15.4 Member Data Documentation

8.15.4.1 LSM303 USU::MinImu::compass

Compass (i.e. Accelerometer and Magnetometer of the IMU)

Definition at line 35 of file minimu.h.

8.15.4.2 L3G USU::MinImu::gyro

Gyroscope of the IMU

Definition at line 36 of file minimu.h.

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

- include/minimu.h
- src/minimu.cpp

8.16 USU::Motor Class Reference

Class which represents a motor.

```
#include <motor.h>
```

Public Member Functions

- Motor (Beagle_GPIO &beagleGpio, Beagle_GPIO::Pins clockwise, Beagle_GPIO::Pins counterClockwise, cPWM &pwm, SetDutyCyle dutyCycle)

  Constructor.
- void setSpeed (int speed)

  Set the speed of the motor in percent.
- int getSpeed () const

  Return the current speed of the motor.

8.16.1 Detailed Description

Class which represents a motor.

It controls 2 digital pins to set motor spin direction and one PWM channel to set motor speed.

Definition at line 37 of file motor.h.

8.16.2 Constructor & Destructor Documentation

8.16.2.1 Motor::Motor (Beagle_GPIO & *beagleGpio*, Beagle_GPIO::Pins *clockwise*, Beagle_GPIO::Pins *counterClockwise*, cPWM & *pwm*, SetDutyCyle *dutyCycle*)

Constructor.

Parameters

beagleGpio	Reference to a Beagle_GPIO object to set the pins
clockwise	First pin needed to set motor direction
counter-Clockwise	Second pin needed to set motor direction
pwm	Reference to the cPWM-object, which controls the PWM
dutyCycle	Function to set the dutyCycle of the PWM-channel assigned to the motor

Definition at line 14 of file motor.cpp.

8.16.3 Member Function Documentation

8.16.3.1 int USU::Motor::getSpeed ( ) const [inline]

Return the current speed of the motor.

Returns

int current Speed of the motor

Definition at line 63 of file motor.h.

8.16.3.2 void Motor::setSpeed (int *speed*)

Set the speed of the motor in percent.

Parameters

speed	desired motor speed (-100, 100)

Definition at line 29 of file motor.cpp.

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

- include/motor.h
- src/motor.cpp

8.17 USU::MotorControl Class Reference

Represents the class for motor control.

```
#include <motorcontrol.h>
```

Public Member Functions

- MotorControl (const char ∗i2cDevice="/dev/i2c-3")

  Constructor of the class.
- virtual ∼MotorControl ()
- void calculateControlResponse (Quaternion state)

  Calculate the control response from the current state estimate.
- void controlFromGyro (const Eigen::Vector3f &gyro)

  Uses a simple algorithm to control the speed only from gyro data.
- void setMotor (int motor, int dutyCycle)

  For testing: sets the speed of a motor.
- void getAnalog (int motor, float &aOut1, float &aOut2)

  For testing: returns the Analog measurements of a motor.
- void getAnalogs (float ∗aOut1, float ∗aOut2)

  For testing: returns the Analog measurements of all motors.
- void getDutyCycles (int ∗dc)

  For testing: returns the dutycycles of all motors.
- float getPGain () const
- void setPGain (float value)
- Eigen::Vector3f getSetValue () const
- void setSetValue (const Eigen::Vector3f value)

8.17.1 Detailed Description

Represents the class for motor control.

It initializes the interface to the 4 motors. It receives the last system state estimate from the Kalman filter, calculates the appropiate control response and sets the speed (duty cycle) of the motors.

TODO: Get the desired state from ground station to calculate the control response.

Definition at line 38 of file motorcontrol.h.

8.17.2 Constructor & Destructor Documentation

8.17.2.1 MotorControl::MotorControl (const char ∗ *i2cDevice* = `"/dev/i2c-3"`)

Constructor of the class.

Initializes the underlying GPIO-class, the PWMs, the 4 Motors and the ADC.

Parameters

i2cDevice	name of the i2cDevice of the ADC

Definition at line 18 of file motorcontrol.cpp.

8.17.2.2 MotorControl::∼MotorControl ( ) [virtual]

Definition at line 34 of file motorcontrol.cpp.

8.17.3 Member Function Documentation

8.17.3.1 void MotorControl::calculateControlResponse (Quaternion *state*)

Calculate the control response from the current state estimate.

TODO: Doesn't do anything at the moment

Parameters

state	the current state estimate from the IMU

TODO: Make some control magic

[...]

Definition at line 42 of file motorcontrol.cpp.

8.17.3.2 void MotorControl::controlFromGyro (const Eigen::Vector3f & *gyro*)

Uses a simple algorithm to control the speed only from gyro data.

Parameters

gyro	Vector with the current angular rates

Definition at line 49 of file motorcontrol.cpp.

8.17.3.3 void MotorControl::getAnalog (int *motor,* float & *aOut1,* float & *aOut2*)

For testing: returns the Analog measurements of a motor.

Parameters

motor	which motor [0..3]
aOut1	reference to a variable to store the first analog measurement
aOut2	reference to a variable to store the second analog measurement

Definition at line 74 of file motorcontrol.cpp.

8.17.3.4 void MotorControl::getAnalogs (float ∗ *aOut1,* float ∗ *aOut2*)

For testing: returns the Analog measurements of all motors.

Parameters

aOut1	Float array to store the first analog measurement of each motor
aOut2	Float array to store the second analog measurement of each motor

Definition at line 80 of file motorcontrol.cpp.

8.17.3.5 void **MotorControl::getDutyCycles** (int ∗ *dc*)

For testing: returns the dutycycles of all motors.

Parameters

dc	Int array to store the duty cycle of each motor

Definition at line 92 of file motorcontrol.cpp.

8.17.3.6 float **MotorControl::getPGain** ( ) const

Definition at line 99 of file motorcontrol.cpp.

8.17.3.7 Eigen::Vector3f **MotorControl::getSetValue** ( ) const

Definition at line 108 of file motorcontrol.cpp.

8.17.3.8 void **MotorControl::setMotor** (int *motor,* int *dutyCycle*)

For testing: sets the speed of a motor.

Parameters

motor	which motor [0..3]
dutyCycle	which speed [-100..100]

Definition at line 69 of file motorcontrol.cpp.

8.17.3.9 void **MotorControl::setPGain** (float *value*)

Definition at line 104 of file motorcontrol.cpp.

8.17.3.10 void **MotorControl::setSetValue** (const Eigen::Vector3f *value*)

Definition at line 113 of file motorcontrol.cpp.

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

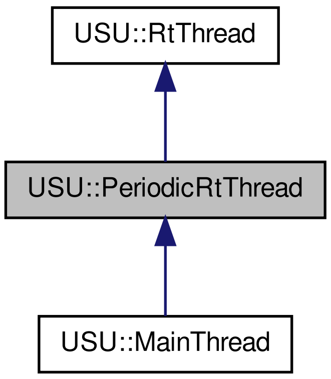
- include/motorcontrol.h
- src/motorcontrol.cpp

8.18 USU::PeriodicRtThread Class Reference

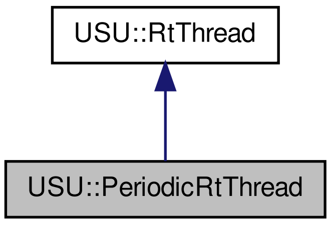
TODO: Make some proper exceptions.

```
#include <periodicrtthread.h>
```

Inheritance diagram for USU::PeriodicRtThread:



Collaboration diagram for USU::PeriodicRtThread:



Public Member Functions

- PeriodicRtThread (int priority=0, unsigned int period_us=1000000)

  Creates the PeriodicRtThread object.
- virtual void run ()=0

  Actual method of the thread is running.

Protected Member Functions

- void makeThreadPeriodic ()

  Registers the Periodic timer.
- void waitPeriod ()

  Blocks the thread until the next timer event.

8.18.1 Detailed Description

TODO: Make some proper exceptions.

Abstract wrapper class for a periodic thread usign the pthread library with RT-priority

Based on RtThread this class uses pthread underneath but creates a periodic timer event it can wait for in a (forever) loop. This is more accurate than the use of nanosleep as the execution time of the loop will not be taken into account. It is therefore designed for periodic work where high accuracy is desired.

Definition at line 32 of file periodicrtthread.h.

8.18.2 Constructor & Destructor Documentation

8.18.2.1 PeriodicRtThread::PeriodicRtThread (int *priority* = `0`, unsigned int *period_us* = `1000000`)

Creates the PeriodicRtThread object.

Calls the constructor of the parent RtThread and registers the periodic timer

Parameters

priority	the Priority of the Thread (Linux: 1..99)
period_us	Period of the thread in us

Definition at line 22 of file periodicrtthread.cpp.

8.18.3 Member Function Documentation

8.18.3.1 void PeriodicRtThread::makeThreadPeriodic ( ) `[protected]`

Registers the Periodic timer.

TODO: create exception

Definition at line 29 of file periodicrtthread.cpp.

8.18.3.2 virtual void USU::PeriodicRtThread::run ( ) `[pure virtual]`

Actual method of the thread is running.

Every child class has to implement this function in order to do some threaded work.

Implements USU::RtThread.

Implemented in USU::MainThread.

8.18.3.3 void PeriodicRtThread::waitPeriod ( ) [protected]

Blocks the thread until the next timer event.

Waits the remaining time until the next timer event happens. Thus waitTime = mPeriod-_us - runtime since last timer event

Definition at line 56 of file periodicrtthread.cpp.

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

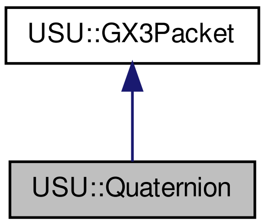
- include/periodicrtthread.h
- src/periodicrtthread.cpp

8.19 USU::Quaternion Class Reference

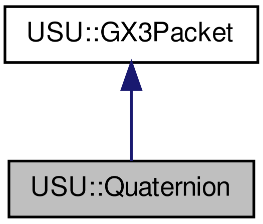
Representation for receiving the Quaternion representation from the IMU.

`#include <messages.h>`

Inheritance diagram for USU::Quaternion:



Collaboration diagram for USU::Quaternion:



Public Types

- enum { size = 23 }

Public Member Functions

- Quaternion ()

  Creates an empty packet object.
- bool readFromSerial (SerialPort &serialPort)

  Read the information for the structure from the SerialPort.
- virtual void print (std::ostream &os) const

  Print the stored information to ostream object.

Public Attributes

- quaternion quat
- unsigned int timer

8.19.1 Detailed Description

Representation for receiving the Quaternion representation from the IMU.

The class will return a Quaternion from the Eigen library

Definition at line 320 of file messages.h.

8.19.2 Member Enumeration Documentation

8.19.2.1 anonymous enum

Enumerator:

size

Definition at line 367 of file messages.h.

8.19.3 Constructor & Destructor Documentation

8.19.3.1 USU::Quaternion::Quaternion ( ) `[inline]`

Creates an empty packet object.

Definition at line 326 of file messages.h.

8.19.4 Member Function Documentation

8.19.4.1 virtual void USU::Quaternion::print (std::ostream & *os*) const `[inline, virtual]`

Print the stored information to ostream object.

quaternion = w + i$*$x + j$*$y + k$*$z

Format: timestamp,w,x,y,z

Parameters

os	

Implements USU::GX3Packet.

Definition at line 358 of file messages.h.

8.19.4.2 bool USU::Quaternion::readFromSerial (SerialPort & *serialPort*) `[inline, virtual]`

Read the information for the structure from the SerialPort.

Parameters

serialPort	serialPort object from libserial

Returns

bool true if reading (and checksum) was successful, false otherwise

Implements USU::GX3Packet.

Definition at line 328 of file messages.h.

8.19.5 Member Data Documentation

8.19.5.1 quaternion USU::Quaternion::quat

Eigen::Quaternionf representing the Orientation of the IMU

Definition at line 363 of file messages.h.

8.19.5.2 unsigned int USU::Quaternion::timer

The value of the timestamp for the package

Definition at line 365 of file messages.h.

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

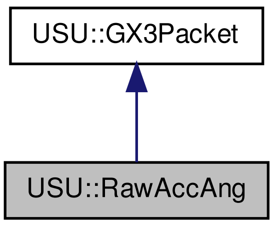
- include/messages.h

8.20 USU::RawAccAng Class Reference

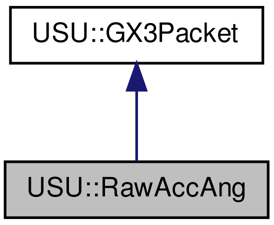
Representation for receiving (raw) acceleration & angular rate packets.

```
#include <messages.h>
```

Inheritance diagram for USU::RawAccAng:



Collaboration diagram for USU::RawAccAng:



Public Types

- enum { size = 31 }

Public Member Functions

- RawAccAng ()

  Creates an empty packet object.
- bool readFromSerial (SerialPort &serialPort)

  Read the information for the structure from the SerialPort.
- virtual void print (std::ostream &os) const

  Print the stored information to ostream object.

Public Attributes

- vector acc
- vector gyro
- unsigned int timer

8.20.1 Detailed Description

Representation for receiving (raw) acceleration & angular rate packets.

This class can be used with the commands for raw acceleration and angular rates and acceleration and angular rate. For the latter the units are:

- acceleration: g
- angular rate: rad/s For the units of the raw values see the protocol data sheet.

Definition at line 191 of file messages.h.

8.20.2 Member Enumeration Documentation

8.20.2.1 anonymous enum

Enumerator:

size

Definition at line 239 of file messages.h.

8.20.3 Constructor & Destructor Documentation

8.20.3.1 USU::RawAccAng::RawAccAng ( ) [inline]

Creates an empty packet object.

Definition at line 197 of file messages.h.

8.20.4 Member Function Documentation

8.20.4.1 virtual void USU::RawAccAng::print (std::ostream & *os*) const [inline, virtual]

Print the stored information to ostream object.

Format: timestamp,accX,accY,accZ,gyroX,gyroY,gyroZ

Parameters

os	

Implements USU::GX3Packet.

Definition at line 228 of file messages.h.

8.20.4.2 bool USU::RawAccAng::readFromSerial (SerialPort & *serialPort*) [inline, virtual]

Read the information for the structure from the SerialPort.

Parameters

serialPort	serialPort object from libserial

Returns

bool true if reading (and checksum) was successful, false otherwise

Implements USU::GX3Packet.

Definition at line 199 of file messages.h.

8.20.5 Member Data Documentation

8.20.5.1 vector USU::RawAccAng::acc

Vector containing the accelerometer data

Definition at line 234 of file messages.h.

8.20.5.2 vector USU::RawAccAng::gyro

Vector containing the gyroscope (angular rate) data

Definition at line 235 of file messages.h.

8.20.5.3 unsigned int USU::RawAccAng::timer

The value of the timestamp for the package

Definition at line 237 of file messages.h.

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

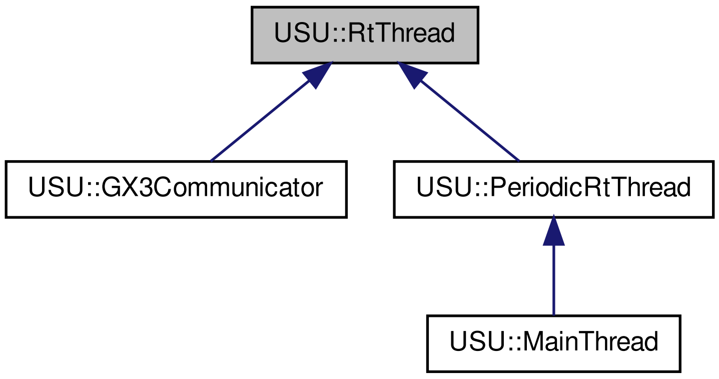
- include/messages.h

8.21 USU::RtThread Class Reference

Abstract wrapper class for the pthread library with RT-priority.

`#include <RtThread.h>`

Inheritance diagram for USU::RtThread:



Public Member Functions

- RtThread (int priority=0)

  Creates the RtThread object.
- virtual ∼RtThread ()

  Destructor of the RtThread object.
- pthread_t getThreadId () const

  Return the pthread handle.
- int getPriority () const

  Returns the priority of the thread.
- void start (void ∗args=NULL)

  Creates and starts the pthread.
- bool join (int timeout_ms=0)

  Waits for the thread to join.
- virtual void run ()=0

  Actual method of the thread is running.

Static Protected Member Functions

- static void ∗ exec (void ∗thr)

  Function passed to pthread_create, do not call manually!

Protected Attributes

- pthread_t mId
- bool mStarted
- void ∗ mArgs

8.21.1 Detailed Description

Abstract wrapper class for the pthread library with RT-priority.

This class is a thin wrapper for the pthread library. Inherited classes need to implement the run function with the tasks for the thread. The thread will run with the SCHED_F-IFO-scheduler at the set priority. Therefore root rights are necessary for changing the scheduling policy.

3

Definition at line 32 of file RtThread.h.

8.21.2 Constructor & Destructor Documentation

8.21.2.1 RtThread::RtThread (int *priority* = 0)

Creates the RtThread object.

Prepares the Attribute object which is passed to pthread_create later.

Parameters

priority	the Priority of the Thread (Linux: 1..99)

Definition at line 19 of file RtThread.cpp.

8.21.2.2 RtThread::∼RtThread ( ) `[virtual]`

Destructor of the RtThread object.

Waits for the thread to join (if not already) and releases the Attributes object.

Definition at line 62 of file RtThread.cpp.

8.21.3 Member Function Documentation

8.21.3.1 void ∗ RtThread::exec (void ∗ *thr*) `[static, protected]`

Function passed to pthread_create, do not call manually!

This function builds the interface to the pthread library. Only purpose is to be compatible to pthread_create, as it will immediately call run of this class.

Parameters

thr	pointer to this instance of the class.

Definition at line 141 of file RtThread.cpp.

8.21.3.2 int RtThread::getPriority ( ) const `[inline]`

Returns the priority of the thread.

Returns

int priority

Definition at line 84 of file RtThread.cpp.

8.21.3.3 pthread_t RtThread::getThreadId ( ) const `[inline]`

Return the pthread handle.

Returns

pthread_t the thread handle of the last started pthread or -1 (if no pthread was started)

Definition at line 78 of file RtThread.cpp.

8.21.3.4 bool RtThread::join (int *timeout_ms* = `0`)

Waits for the thread to join.

Parameters

timeout_ms	timeout in ms (optional). 0 means no timeout

Returns

bool returns true if thread joined successfully and false if error occured

Definition at line 110 of file RtThread.cpp.

8.21.3.5 virtual void USU::RtThread::run ( ) [pure virtual]

Actual method of the thread is running.

Every child class has to implement this function in order to do some threaded work.

Implemented in USU::PeriodicRtThread, USU::GX3Communicator, and USU::MainThread.

8.21.3.6 void RtThread::start (void ∗ *args* = NULL)

Creates and starts the pthread.

Creates the pthread with the desired attributes.

Parameters

args	optional arguments for the thread

Definition at line 89 of file RtThread.cpp.

8.21.4 Member Data Documentation

8.21.4.1 void∗ USU::RtThread::mArgs [protected]

Arguments which can be passed to a certain thread thread

Definition at line 45 of file RtThread.h.

8.21.4.2 pthread_t USU::RtThread::mId [protected]

The thread handle

Definition at line 43 of file RtThread.h.

8.21.4.3 bool USU::RtThread::mStarted [protected]

Keeps the status of the thread TODO: Useful??

Definition at line 44 of file RtThread.h.

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

- include/RtThread.h

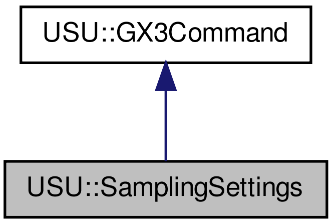
- src/RtThread.cpp

8.22 USU::SamplingSettings Class Reference

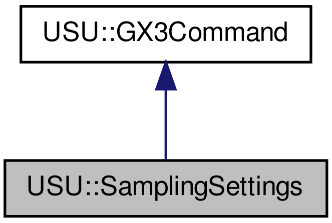
Represents the "Sampling Settings" command.

`#include <messages.h>`

Inheritance diagram for USU::SamplingSettings:



Collaboration diagram for USU::SamplingSettings:



Public Types

- enum FunctionSelector { ReturnOnly = 0, Change = 1, ChangeAndSave = 2, -ChangeWithoutReply = 3 }

  Sets the function Selector.

- enum DataConditioning { FlagCalcOrientation = 0x01, FlagEnableConing-Sculling = 0x02, FlagDefault = 0x03, FlagFloatLittleEndian = 0x10, Flag-SuppressNaN = 0x20, FlagFiniteSizeCorrection = 0x40, FlagDisableMag = 0x100, FlagDisableMagNorthComp = 0x400, FlagDisableGravComp = 0x800, FlagEnableQuaternion = 0x1000 }

  Flags for the Data conditioning.
- enum { size = 20, responseSize = 19 }

Public Member Functions

- SamplingSettings (FunctionSelector funSel, uint16_t samplingPeriod_ms=10, uint16_t dataCondFlags=SamplingSettings::FlagDefault, uint8_t gyroAcc-Filter=15, uint8_t magFilter=17, uint16_t upCompensation=10, uint16_t north-Compensation=10, uint8_t magPower=0)

  Creates the command.
- bool sendCommand (SerialPort &serialPort)
- bool checkResponse (uint8_t ∗buffer)

  Checks if the response to this command has the correct setup.

Public Attributes

- uint8_t mCommand [size]

8.22.1 Detailed Description

Represents the "Sampling Settings" command.

Definition at line 512 of file messages.h.

8.22.2 Member Enumeration Documentation

8.22.2.1 anonymous enum

Enumerator:

size

responseSize

Definition at line 628 of file messages.h.

8.22.2.2 enum USU::SamplingSettings::DataConditioning

Flags for the Data conditioning.

Sets the bits for Data conditioning bytes. Combine multiple flags using the "or" operator ("|")

Enumerator:

FlagCalcOrientation

FlagEnableConingSculling

FlagDefault

FlagFloatLittleEndian

FlagSuppressNaN

FlagFiniteSizeCorrection

FlagDisableMag

FlagDisableMagNorthComp

FlagDisableGravComp

FlagEnableQuaternion

Definition at line 537 of file messages.h.

8.22.2.3 enum USU::SamplingSettings::FunctionSelector

Sets the function Selector.

The function selector has 4 states:

- ReturnOnly: Does not change the Sampling Settings, only returns the current state
- Change: Set new Sampling settings, but do not store them in non-volatile memory (will be reset after shutdown)
- ChangeAndSave: Set new Sampling Settings and store them in non-volatile memory (will be permanent)
- ChangeWithoutReply: As Change but no response is sent

Enumerator:

ReturnOnly

Change

ChangeAndSave

ChangeWithoutReply

Definition at line 526 of file messages.h.

8.22.3 Constructor & Destructor Documentation

8.22.3.1 USU::SamplingSettings::SamplingSettings (FunctionSelector *funSel*, uint16_t *samplingPeriod_ms* = `10`, uint16_t *dataCondFlags* = SamplingSettings::FlagDefault, uint8_t *gyroAccFilter* = `15`, uint8_t *magFilter* = `17`, uint16_t *upCompensation* = `10`, uint16_t *northCompensation* = `10`, uint8_t *magPower* = `0`) `[inline]`

Creates the command.

Allocates a buffer for the byte commands. Sets the static bytes and fills the settings bytes based on the passed parameters.

Parameters

funSel	Sets the functions selector
sampling-Period_ms	Sets the sampling period in ms (1 to 1000)
dataCond-Flags	Sets general behaviour of the 3DM; use DataConditioning-flags
gyroAcc-Filter	Sets the filter value for the gyro and accelerometer
magFilter	Sets the filter value for the magnetometer
up-Compensation	Sets the time for up compensation
north-Compensation	Sets the time for north compensation
magPower	Sets the Power state

Definition at line 567 of file messages.h.

8.22.4 Member Function Documentation

8.22.4.1 bool **USU::SamplingSettings::checkResponse** (uint8_t ∗ *buffer*) [inline, virtual]

Checks if the response to this command has the correct setup.

Parameters

buffer	pointer to the byte array containing the response from the 3DM

Returns

bool true if the response is correct, false if it suggests an error

Implements USU::GX3Command.

Definition at line 613 of file messages.h.

8.22.4.2 bool **USU::SamplingSettings::sendCommand** (SerialPort & *serialPort*) [inline, virtual]

Implements USU::GX3Command.

Definition at line 596 of file messages.h.

8.22.5 Member Data Documentation

8.22.5.1 uint8_t USU::SamplingSettings::mCommand[size]

Buffer which contains the byte array for the command

Definition at line 629 of file messages.h.

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

- include/messages.h

8.23 USU::ScopedLock Class Reference

Provides a helper class for Scoped Mutexes.

```
#include <Lock.h>
```

Public Member Functions

- ScopedLock (Lock &lock)

  Constructor: will lock the mutex.
- virtual ∼ScopedLock ()

  Destructor: will unlock the mutex.

8.23.1 Detailed Description

Provides a helper class for Scoped Mutexes.

Create this object by passing a reference to a Lock object. It will lock the mutex when created and unlock it when destroyed, i.e. when going out of scope at the end of the "}". Can make it more convenient than manual (un)locking.

TODO: Test if it works correctly with a getter-method

Definition at line 92 of file Lock.h.

8.23.2 Constructor & Destructor Documentation

8.23.2.1 USU::ScopedLock::ScopedLock (Lock & *lock*) `[inline]`

Constructor: will lock the mutex.

Parameters

lock	Reference to the Lock it needs to hold

Definition at line 115 of file Lock.h.

8.23.2.2 USU::ScopedLock::∼ScopedLock ( ) `[inline, virtual]`

Destructor: will unlock the mutex.

Definition at line 122 of file Lock.h.

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

- include/Lock.h

8.24 USU::Semaphore Class Reference

Wrapper class for semaphores.

`#include <semaphore.h>`

Public Member Functions

- Semaphore ()
- virtual ∼Semaphore ()
- void post ()
- void wait ()

  Trys to get the semaphore, blocking.
- bool tryWait ()

  Trys to get the semaphore, non-blocking.

8.24.1 Detailed Description

Wrapper class for semaphores.

Definition at line 27 of file semaphore.h.

8.24.2 Constructor & Destructor Documentation

8.24.2.1 USU::Semaphore::Semaphore ( )

Constructor: Creates the pthread-Semaphore

Definition at line 63 of file semaphore.h.

8.24.2.2 USU::Semaphore::∼Semaphore ( ) `[virtual]`

Destructor: Frees the pthread-Semaphore

Definition at line 72 of file semaphore.h.

8.24.3 Member Function Documentation

8.24.3.1 void USU::Semaphore::post ( ) `[inline]`

Increases the semaphore by 1

Definition at line 82 of file semaphore.h.

8.24.3.2 bool USU::Semaphore::tryWait ( ) [inline]

Trys to get the semaphore, non-blocking.

Takes the semaphore by decreasing the counter by 1, will return if the counter = 0.

Returns

bool false if semaphore was empty, true if semaphore was successfully acquired

Definition at line 94 of file semaphore.h.

8.24.3.3 void USU::Semaphore::wait ( ) [inline]

Trys to get the semaphore, blocking.

Takes the semaphore by decreasing the counter by 1, will wait for the semaphore to be given if the counter = 0.

Definition at line 88 of file semaphore.h.

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

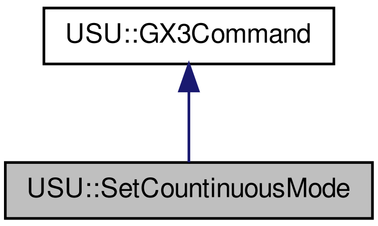
- include/semaphore.h

8.25 USU::SetCountinuousMode Class Reference

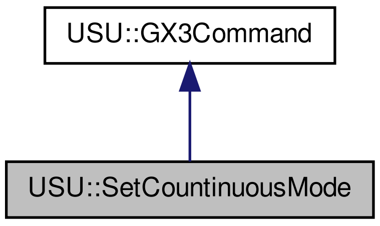
Represents the "Set continuous mode" command.

`#include <messages.h>`

Inheritance diagram for USU::SetCountinuousMode:



Collaboration diagram for USU::SetCountinuousMode:



Public Types

- enum { size = 4, responseSize = 8 }

Public Member Functions

- SetCountinuousMode (uint8_t CommandByte=0)

    Creates the command.
- bool sendCommand (SerialPort &serialPort)
- bool checkResponse (uint8_t ∗buffer)

    Checks if the response to this command has the correct setup.

Public Attributes

- uint8_t mCommand [size]

8.25.1 Detailed Description

Represents the "Set continuous mode" command.

Definition at line 455 of file messages.h.

8.25.2 Member Enumeration Documentation

8.25.2.1 anonymous enum

Enumerator:

size

responseSize

Definition at line 505 of file messages.h.

8.25.3 Constructor & Destructor Documentation

8.25.3.1 USU::SetCountinuousMode::SetCountinuousMode (uint8_t *CommandByte* = 0) [inline]

Creates the command.

Allocates a buffer for the byte commands. Sets the static bytes and fills the settings bytes based on the passed parameters.

Parameters

Command-Byte	Command code of the command which is to be executed periodically (Default stop continuous mode)

Definition at line 467 of file messages.h.

8.25.4 Member Function Documentation

8.25.4.1 bool USU::SetCountinuousMode::checkResponse (uint8_t ∗ *buffer*) [inline, virtual]

Checks if the response to this command has the correct setup.

Parameters

buffer	pointer to the byte array containing the response from the 3DM

Returns

bool true if the response is correct, false if it suggests an error

Implements USU::GX3Command.

Definition at line 492 of file messages.h.

8.25.4.2 bool USU::SetCountinuousMode::sendCommand (SerialPort & *serialPort*) [inline, virtual]

Implements USU::GX3Command.

Definition at line 475 of file messages.h.

8.25.5 Member Data Documentation

8.25.5.1 uint8_t USU::SetCountinuousMode::mCommand[size]

Buffer which contains the byte array for the command

Definition at line 506 of file messages.h.

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

- include/messages.h

8.26 USU::SharedQueue< T > Class Template Reference

Wrapper class to make std::queue thread safe.

```
#include <sharedqueue.h>
```

Public Member Functions

- void push (const T &newElement)

  Constructor, creates an empty queue.
- void pop ()

  Destroys the first (oldest) element in the queue.
- T & front ()

  Returns a reference to the first (oldest) element in the queue.
- bool isEmpty ()

  Indicates if the queue is empty.
- int size ()

8.26.1 Detailed Description

template<class T>class USU::SharedQueue< T >

Wrapper class to make std::queue thread safe.

Protects the push, pop and front access from thread using a mutex. It can only handle one reader and one writer thread at a time. Multiple reader threads could produce race conditions!!!

3

Definition at line 35 of file sharedqueue.h.

8.26.2 Member Function Documentation

8.26.2.1 template<class T> T& USU::SharedQueue< T >::front ( ) [inline]

Returns a reference to the first (oldest) element in the queue.

Takes a mutex before accesing the first element.

Returns

T

Definition at line 77 of file sharedqueue.h.

8.26.2.2 template<class T> bool USU::SharedQueue< T >::isEmpty ( ) [inline]

Indicates if the queue is empty.

Returns

bool true if empty, false otherwise

Definition at line 88 of file sharedqueue.h.

8.26.2.3 template<class T> void USU::SharedQueue< T >::pop ( ) [inline]

Destroys the first (oldest) element in the queue.

Takes mutex before the write operation. Calls the destroy operator of the current front-element.

Definition at line 64 of file sharedqueue.h.

8.26.2.4 template<class T> void USU::SharedQueue< T >::push (const T & *newElement*) [inline]

Constructor, creates an empty queue.

Adds a new element to the back of the queue

Takes the mutex before the write operation.

Parameters

newElement	the element to be added

Definition at line 51 of file sharedqueue.h.

8.26.2.5 template<class T> int USU::SharedQueue< T >::size ( ) [inline]

Definition at line 94 of file sharedqueue.h.

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

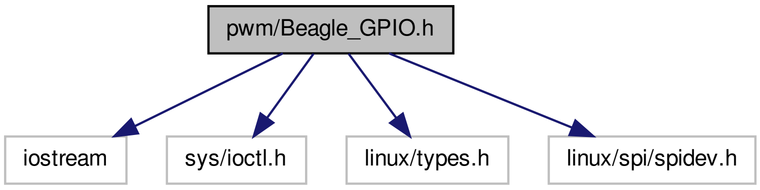
- include/sharedqueue.h

9 File Documentation

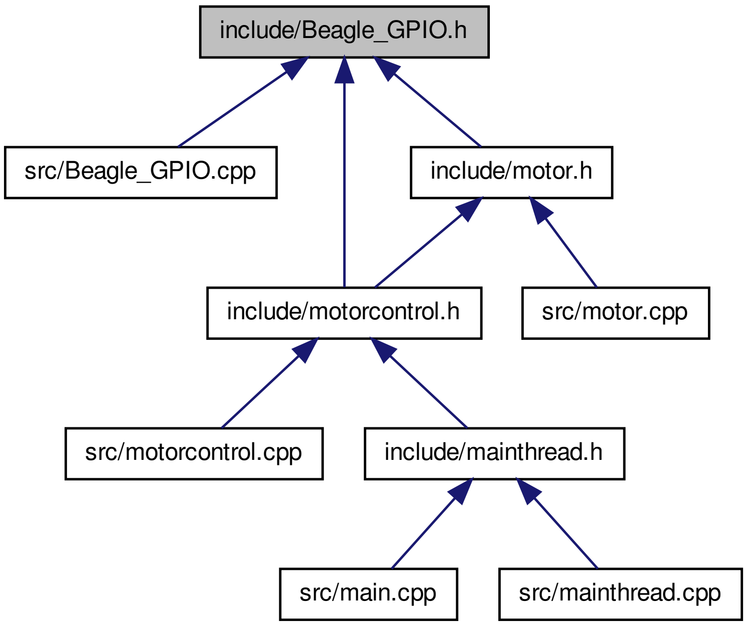
9.1 include/Beagle_GPIO.h File Reference

`#include <iostream> #include <sys/ioctl.h> #include <linux/types.-`

`h>` `#include <linux/spi/spidev.h>` Include dependency graph for Beagle-_GPIO.h:



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



Classes

- class Beagle_GPIO

Wrapper class to access the GPIOs of the BeagleBone.

Defines

- #define GPIO_ERROR(msg) std::cout << "[GPIO] Error : " << msg << std::endl;
- #define BEAGLE_GPIO_DEBUG
- #define GPIO_PRINT(msg) std::cout << "[GPIO] : " << msg << std::endl;
- #define gp_assert(condition)

9.1.1 Define Documentation

9.1.1.1 #define BEAGLE_GPIO_DEBUG

Definition at line 29 of file Beagle_GPIO.h.

9.1.1.2 #define gp_assert(*condition*)

Value:

```
if (!(condition))                \
                                 {                                             \
                                               GPIO_ERROR( "Assert Failed in file
       '" << __FILE__ << "' on line " << __LINE__ );                \
                                               exit(0);        \
                                 }
```

Definition at line 32 of file Beagle_GPIO.h.

9.1.1.3 #define GPIO_ERROR(*msg*) std::cout << "[GPIO] Error : " << msg << std::endl;

Definition at line 27 of file Beagle_GPIO.h.

9.1.1.4 #define GPIO_PRINT(*msg*) std::cout << "[GPIO] : " << msg << std::endl;

Definition at line 31 of file Beagle_GPIO.h.

9.2 include/Beagle_GPIO.h

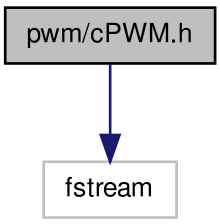
```
/******************************
 ** Beagle Bone GPIO Library **
 **                          **
 **      Francois Sugny      **
 **         01/07/12         **
 **                          **
 **          v1.0            **
 ******************************/

//=======================================================
//=======================================================

#ifndef beagle_gpio_hh
#define beagle_gpio_hh
```

```

//=======================================================
//=======================================================

#include <iostream>
#include <sys/ioctl.h>
#include <linux/types.h>
#include <linux/spi/spidev.h>

//=======================================================
//=======================================================

#define GPIO_ERROR(msg)        std::cout << "[GPIO] Error : " << msg <<
 std::endl;

#define BEAGLE_GPIO_DEBUG
#ifdef BEAGLE_GPIO_DEBUG
               #define GPIO_PRINT(msg)       std::cout << "[GPIO] : " << msg <<
 std::endl;
    #define gp_assert( condition )              \
                              if (!(condition))               \
                              {                                              \
                                             GPIO_ERROR( "Assert Failed in file
 '" << __FILE__ << "' on line " << __LINE__ );            \
                                             exit(0);        \
                              }

#else
               #define GPIO_PRINT(msg)
    #define gp_assert( condition )
#endif


//=======================================================
//=======================================================

class Beagle_GPIO
{
public:
               // Return status
               typedef enum
               {
                              kFail                          = 0,
                              kSuccess       = 1
               } Beagle_GPIO_Status;

               // Beagle Bone GPIO Register Offsets
               enum
               {
                              kREVISION                      = 0x0,
                              kSYSCONFIG                     = 0x10,
                              kIRQSTATUS_RAW_0               = 0x24,
                              kIRQSTATUS_RAW_1               = 0x28,
                              kIRQSTATUS_0                   = 0x2C,
                              kIRQSTATUS_1                   = 0x30,
                              kIRQSTATUS_SET_0               = 0x34,
                              kIRQSTATUS_SET_1               = 0x38,
                              kIRQSTATUS_CLR_0               = 0x3C,
                              kIRQSTATUS_CLR_1               = 0x40,
                              kIRQWAKEN_0                    = 0x44,
                              kIRQWAKEN_1                    = 0x48,
                              kSYSSTATUS                     = 0x114,
                              kCTRL                                         =
0x130,
                              kOE                                           =
0x134,
                              kDATAIN                                       =
0x138,
                              kDATAOUT                       = 0x13C,
                              kLEVELDETECT0                  = 0x140,
                              kLEVELDETECT1                  = 0x144,
                              kRISINGDETECT                  = 0x148,
                              kFALLINGDETECT                 = 0x14C,
                              kDEBOUNCEENABLE                               =
0x150,
                              kDEBOUNCINGTIME                               =
0x154,
                              kCLEARDATAOUT                  = 0x190,
                              kSETDATAOUT                    = 0x194
               } Beagle_GPIO_Registers;

               // Input/Output pin mode
               typedef enum
               {
                              kINPUT         = 0,
                              kOUTPUT = 1
               } Beagle_GPIO_Direction;

               // GPIO Pins
     enum Pins
               {
                              P8_1,  P8_2,  P8_3,  P8_4,  P8_5,
                              P8_6,  P8_7,  P8_8,  P8_9,  P8_10,
                              P8_11, P8_12, P8_13, P8_14, P8_15,
                              P8_16, P8_17, P8_18, P8_19, P8_20,
                              P8_21, P8_22, P8_23, P8_24, P8_25,
                              P8_26, P8_27, P8_28, P8_29, P8_30,
                              P8_31, P8_32, P8_33, P8_34, P8_35,
                              P8_36, P8_37, P8_38, P8_39, P8_40,
                              P8_41, P8_42, P8_43, P8_44, P8_45,
                              P8_46,
                              P9_1,  P9_2,  P9_3,  P9_4,  P9_5,
                              P9_6,  P9_7,  P9_8,  P9_9,  P9_10,
                              P9_11, P9_12, P9_13, P9_14, P9_15,
                              P9_16, P9_17, P9_18, P9_19, P9_20,
                              P9_21, P9_22, P9_23, P9_24, P9_25,
                              P9_26, P9_27, P9_28, P9_29, P9_30,
                              P9_31, P9_32, P9_33, P9_34, P9_35,
                              P9_36, P9_37, P9_38, P9_39, P9_40,
                              P9_41, P9_42, P9_43, P9_44, P9_45,
                              P9_46
               } GPIO_Pins;

               // IO Banks for GPIOs
               static const int GPIO_Pin_Bank[];

               // Pin Id for GPIOs
               static const int GPIO_Pin_Id[];

               // Pad Control Register
               static const unsigned long GPIO_Pad_Control[];

               // Base address of Control Module Registers
               static const unsigned long GPIO_Control_Module_Registers;

               // Base addresses of GPIO Modules
               static const unsigned long GPIO_Base[];

public:
               Beagle_GPIO();
               ~Beagle_GPIO();

public:
               // Configure pin as input/output
               Beagle_GPIO_Status configurePin( unsigned short _pin,
Beagle_GPIO_Direction _direction );

               // Enable/Disable interrupts for the pin
               Beagle_GPIO_Status enablePinInterrupts( unsigned short _pin,
bool _enable );

               // Write a value to a pin
               Beagle_GPIO_Status writePin( unsigned short _pin, unsigned char
_value );

               // Read a value from a pin
               unsigned char readPin( unsigned short _pin );

```

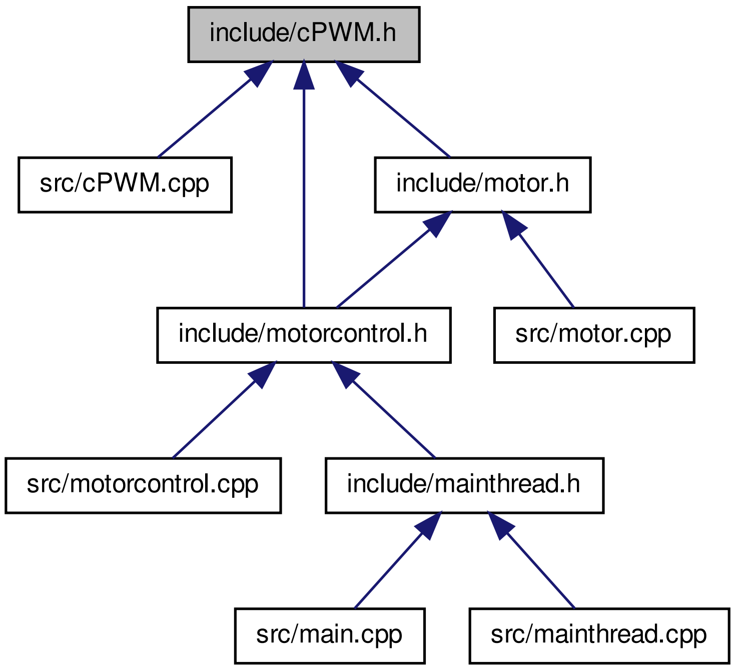
```
                // Open SPI Channel
                void openSPI( unsigned char _mode=0,
                                      unsigned char _bits=8,
                                      unsigned long _speed=4800000,
                                      unsigned short _delay=0 );

                // Close SPI Channel
                void closeSPI();

                // Send SPI Buffer
                void sendSPIBuffer( unsigned long buffer, int size );

                // Is this Module active ?
                bool isActive() { return m_active; }

private:
                bool                                          m_active;
                int                                           m_gpio_fd;
                unsigned long *                               m_controlModule;
                unsigned long *               m_gpio[4];

                int                                           m_spi_fd;
                unsigned char *                               m_spi_buffer_rx;
                unsigned char                 m_spi_mode;
                unsigned char                 m_spi_bits;
                unsigned long                 m_spi_speed;
                unsigned short                                m_spi_delay;

                struct spi_ioc_transfer m_spi_ioc_tr;
};

//=======================================================
//=======================================================

#endif

//=======================================================
//=======================================================

```

9.3 include/cPWM.h File Reference

`#include <fstream>` Include dependency graph for cPWM.h:



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



Classes

- class cPWM

  Wrapper class to access the PWM-devices of the BeagleBone.

Defines

- #define SYSFS_EHRPWM_PREFIX "/sys/class/pwm/ehrpwm."
- #define SYSFS_EHRPWM_SUFFIX_A ":0"
- #define SYSFS_EHRPWM_SUFFIX_B ":1"
- #define SYSFS_EHRPWM_DUTY_NS "duty_ns"
- #define SYSFS_EHRPWM_DUTY_PERCENT "duty_percent"
- #define SYSFS_EHRPWM_PERIOD_NS "period_ns"
- #define SYSFS_EHRPWM_PERIOD_FREQ "period_freq"
- #define SYSFS_EHRPWM_POLARITY "polarity"
- #define SYSFS_EHRPWM_RUN "run"
- #define SYSFS_EHRPWM_REQUEST "request"

9.3.1 Detailed Description

Simple C++ class wrapper for beaglebone PWM eHRPWM interface header file

Author

claus Created on: Jun 13, 2012 Author: claus `http://quadrotordiaries.-blogspot.com`

Definition in file cPWM.h.

9.3.2 Define Documentation

9.3.2.1 #define SYSFS_EHRPWM_DUTY_NS "duty_ns"

Definition at line 67 of file cPWM.h.

9.3.2.2 #define SYSFS_EHRPWM_DUTY_PERCENT "duty_percent"

Definition at line 68 of file cPWM.h.

9.3.2.3 #define SYSFS_EHRPWM_PERIOD_FREQ "period_freq"

Definition at line 70 of file cPWM.h.

9.3.2.4 #define SYSFS_EHRPWM_PERIOD_NS "period_ns"

Definition at line 69 of file cPWM.h.

9.3.2.5 #define SYSFS_EHRPWM_POLARITY "polarity"

Definition at line 71 of file cPWM.h.

9.3.2.6 #define SYSFS_EHRPWM_PREFIX "/sys/class/pwm/ehrpwm."

Definition at line 64 of file cPWM.h.

9.3.2.7 #define SYSFS_EHRPWM_REQUEST "request"

Definition at line 73 of file cPWM.h.

9.3.2.8 #define SYSFS_EHRPWM_RUN "run"

Definition at line 72 of file cPWM.h.

9.3.2.9 #define SYSFS_EHRPWM_SUFFIX_A ":0"

Definition at line 65 of file cPWM.h.

9.3.2.10 #define SYSFS_EHRPWM_SUFFIX_B ":1"

Definition at line 66 of file cPWM.h.

9.4 include/cPWM.h

```
// $Id$
// $Log$

#ifndef CPWM_H_
#define CPWM_H_

#include <fstream>

class cPWM {

public:
    enum Polarity
    {
        ActiveHigh,
        ActiveLow
    };

private:
    int id;
    unsigned int dutyA;
    unsigned int dutyB;
    unsigned int period;
    unsigned int freq_Hz;
    enum cPWM::Polarity polarityA;
    enum cPWM::Polarity polarityB;
    int runA;
    int runB;
    /*****************************************
                *
                * sysfs tree:
                *
ehrpwm.0:0
    duty_ns
    period_ns
    polarity
    request
    run
 ehrpwm.0:1 -> ../../devices/platform/omap/ehrpwm.0/pwm/ehrpwm.0:1
    duty_ns
    period_ns
    polarity
    request
    run
                *
std::stringstream sysfs_file;

Define files to match sysfs tree:
                */
                              #define SYSFS_EHRPWM_PREFIX "/sys/class/pwm/
ehrpwm."
                              #define SYSFS_EHRPWM_SUFFIX_A ":0"
                              #define SYSFS_EHRPWM_SUFFIX_B ":1"
        #define SYSFS_EHRPWM_DUTY_NS "duty_ns"
        #define SYSFS_EHRPWM_DUTY_PERCENT "duty_percent"
        #define SYSFS_EHRPWM_PERIOD_NS "period_ns"
        #define SYSFS_EHRPWM_PERIOD_FREQ "period_freq"
                              #define SYSFS_EHRPWM_POLARITY "polarity"
                              #define SYSFS_EHRPWM_RUN "run"
                              #define SYSFS_EHRPWM_REQUEST "request"

        std::ofstream sysfsfid_dutyA_ns;
        std::ofstream sysfsfid_dutyA_percent;
        std::ofstream sysfsfid_dutyB_ns;
        std::ofstream sysfsfid_dutyB_percent;
        std::ofstream sysfsfid_period_ns;
        std::ofstream sysfsfid_period_freq;
                              std::ofstream sysfsfid_polarityA;
                              std::ofstream sysfsfid_runA;
                              std::ofstream sysfsfid_requestA;
                              std::ofstream sysfsfid_polarityB;
                              std::ofstream sysfsfid_runB;
```

```
                               std::ofstream sysfsfid_requestB;

public:

              cPWM(int id);
    virtual ~cPWM();

    void DutyA_ns(unsigned int nanoseconds);
    void DutyA_percent(unsigned int percent);  //TODO: check if floats are
 possible

    void DutyB_ns(unsigned int nanoseconds);
    void DutyB_percent(unsigned int percent); //TODO: check if floats are
 possible

    void Period_ns(unsigned int nanoseconds);
    void Period_freq(unsigned int freq_Hz);

    void PolarityA(cPWM::Polarity polarity);
    void RunA();
    void StopA();
    void PolarityB(cPWM::Polarity polarity);
    void RunB();
    void StopB();
};

#endif /* CPWM_H_ */
```

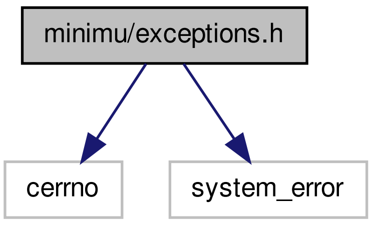
9.5 include/doxygen.h File Reference

9.6 include/doxygen.h

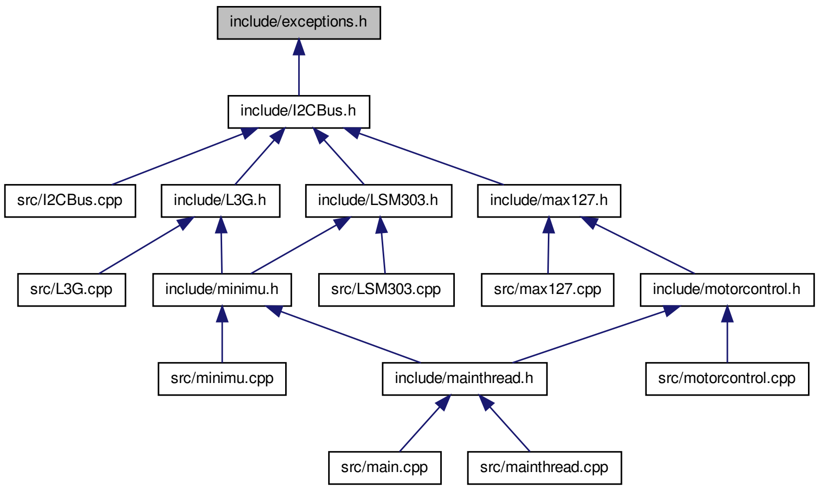
```

```

9.7 include/exceptions.h File Reference

`#include <cerrno>` `#include <system_error>` Include dependency graph for exceptions.h:



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



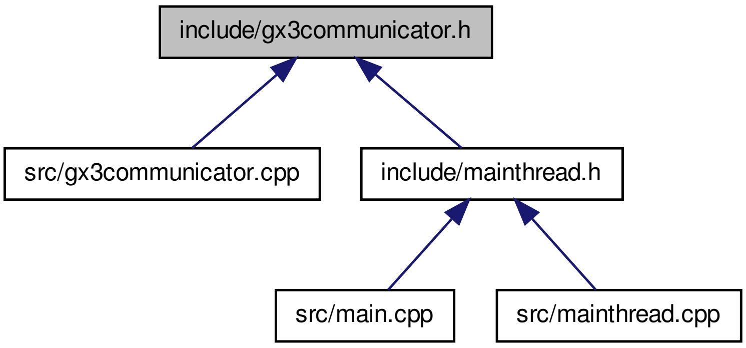
9.8 include/exceptions.h

```
#ifndef _AHRS_EXCEPTIONS_H
#define _AHRS_EXCEPTIONS_H

#include <cerrno>
#include <system_error>

static inline std::system_error posix_error()
{
    return std::system_error(errno, std::system_category());
}

static inline std::system_error posix_error(const char * what)
{
    return std::system_error(errno, std::system_category(), what);
}

#endif
```

9.9 include/gx3communicator.h File Reference

`#include <SerialPort.h>` `#include <memory>` `#include "RtThread.h"` `#include "sharedqueue.h"` `#include "messages.h"`

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



Classes

- class USU::GX3Communicator

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

Typedefs

- typedef std::shared_ptr < GX3Packet > USU::packet_ptr

  Represents the Thread class for communication with the 3DM-GX3-25.

9.9.1 Detailed Description

Contains the thread which handles the communication to the 3DM-GX3-25.

Author

Jan Sommer Created on: Apr 26, 2013

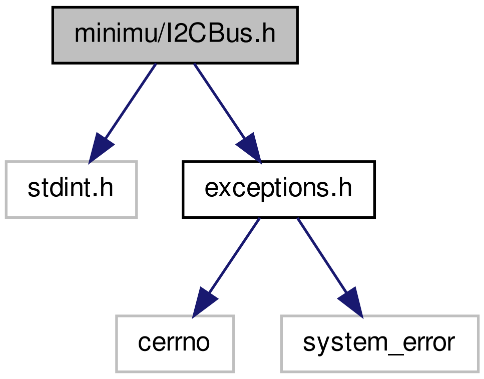
Definition in file gx3communicator.h.

9.10 include/gx3communicator.h

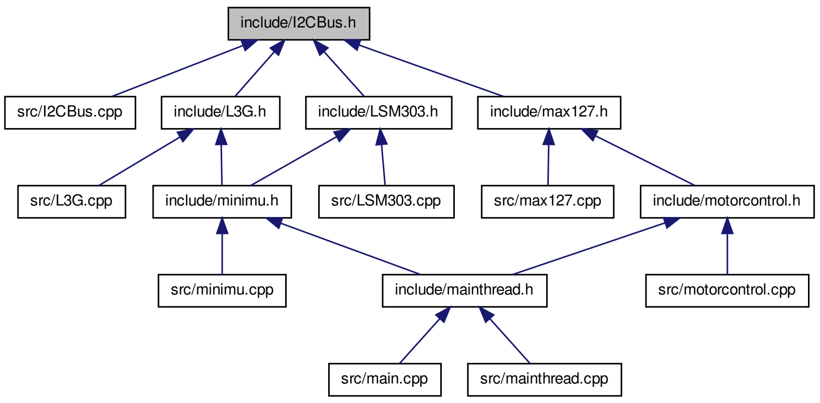
```
#ifndef GX3COMMUNICATOR_H
#define GX3COMMUNICATOR_H

#include<SerialPort.h>
#include<memory>
#include "RtThread.h"
#include "sharedqueue.h"
#include "messages.h"

namespace USU
{

typedef std::shared_ptr<GX3Packet> packet_ptr;

class GX3Communicator : public RtThread
{
public:

    GX3Communicator(int priority, const char* serialDevice,
                    SerialPort::BaudRate baudRate = SerialPort::BAUD_115200);


    void initialize();

    virtual void run();

    void stop() {mKeepRunning = false;}

    void pop() { mQueue.pop();}


    bool isEmpty() {return mQueue.isEmpty(); }

    unsigned size() {return mQueue.size(); }

    packet_ptr &front() { return mQueue.front(); }

private:
    GX3Communicator(const GX3Communicator& thread);
    GX3Communicator& operator=(const GX3Communicator& rhs);
    SerialPort mSerialPort;
    SharedQueue<packet_ptr> mQueue;
    SerialPort::BaudRate mBaudRate;

    volatile bool mKeepRunning;
};

}

#endif // GX3COMMUNICATOR_H
```

9.11 include/I2CBus.h File Reference

`#include <stdint.h>` `#include "exceptions.h"` Include dependency graph for I2CBus.h:



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



Classes

- class I2CBus

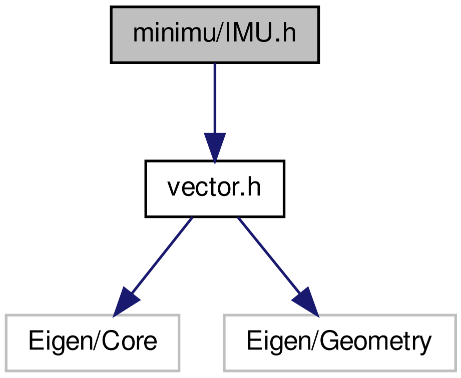
Wrapper class for I2C-bus communication.

9.12 include/I2CBus.h

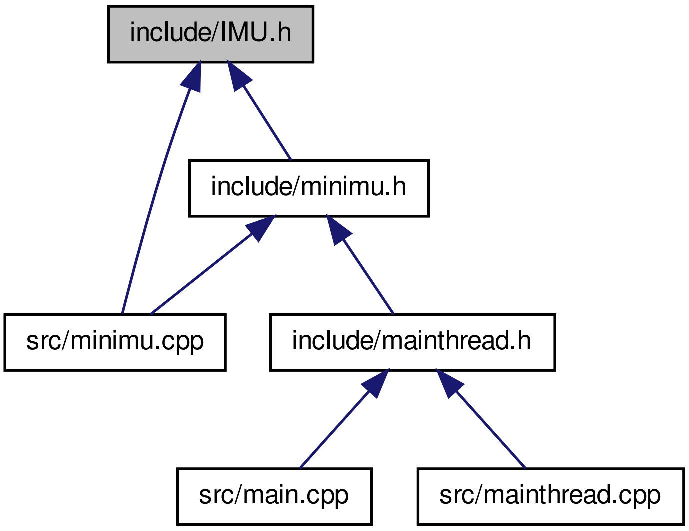
```
#ifndef _I2CBus_h
#define _I2CBus_h

#include <stdint.h>
#include "exceptions.h"


class I2CBus
{
public:
    I2CBus(const char * deviceName);

    ~I2CBus();

    void addressSet(uint8_t address);

    void writeByte(uint8_t command, uint8_t data);

    void writeByte(uint8_t data);

    uint8_t readByte(uint8_t command);

    uint8_t readByte();

    uint16_t readWord(uint8_t command);

    uint16_t readWord();

    int tryReadByte(uint8_t command);

    void readBlock(uint8_t command, uint8_t size, uint8_t * data);

private:
    int fd;
};

#endif
```

9.13 include/IMU.h File Reference

`#include "vector.h"` Include dependency graph for IMU.h:



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



Classes

- class IMU

  Virtual base class for IMU.

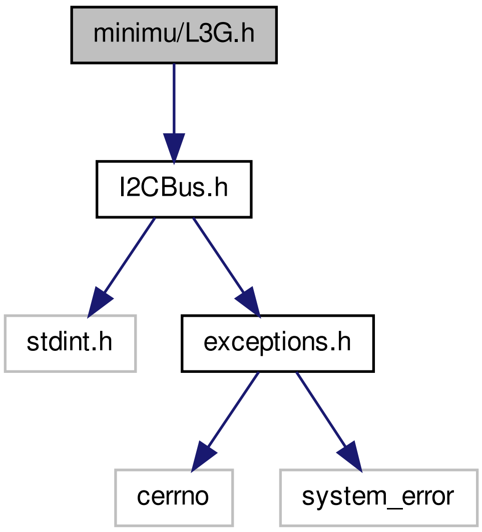
9.14 include/IMU.h

```
#ifndef _IMU_H
#define _IMU_H

#include "vector.h"

class IMU {
public:
    // Scaled readings
    virtual vector readMag() = 0;  // In body coords, scaled to -1..1 range
    virtual vector readAcc() = 0;  // In body coords, with units = g
    virtual vector readGyro() = 0; // In body coords, with units = rad/sec
    void read(){ readAcc(); readMag(); readGyro(); }

    virtual void enable() = 0;
//    virtual void measureOffsets() = 0;
//    virtual void loadCalibration() = 0;

//    vector gyro_offset;
//    matrix calMatrix;
//    vector calOffset;

```

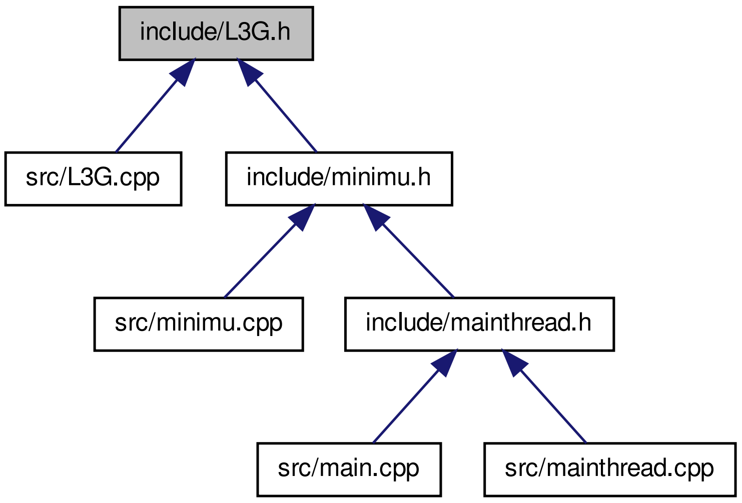
```
    int_vector raw_m, raw_a, raw_g;
};

#endif
```

9.15 include/L3G.h File Reference

`#include "I2CBus.h"` Include dependency graph for L3G.h:



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



Classes

- class L3G

  Class to manage the communication to the L3G gyroscope via the I2C-bus.

Defines

- #define L3G_WHO_AM_I 0x0F
- #define L3G_CTRL_REG1 0x20
- #define L3G_CTRL_REG2 0x21
- #define L3G_CTRL_REG3 0x22
- #define L3G_CTRL_REG4 0x23
- #define L3G_CTRL_REG5 0x24
- #define L3G_REFERENCE 0x25
- #define L3G_OUT_TEMP 0x26
- #define L3G_STATUS_REG 0x27
- #define L3G_OUT_X_L 0x28
- #define L3G_OUT_X_H 0x29
- #define L3G_OUT_Y_L 0x2A
- #define L3G_OUT_Y_H 0x2B
- #define L3G_OUT_Z_L 0x2C

- #define L3G_OUT_Z_H 0x2D
- #define L3G_FIFO_CTRL_REG 0x2E
- #define L3G_FIFO_SRC_REG 0x2F
- #define L3G_INT1_CFG 0x30
- #define L3G_INT1_SRC 0x31
- #define L3G_INT1_THS_XH 0x32
- #define L3G_INT1_THS_XL 0x33
- #define L3G_INT1_THS_YH 0x34
- #define L3G_INT1_THS_YL 0x35
- #define L3G_INT1_THS_ZH 0x36
- #define L3G_INT1_THS_ZL 0x37
- #define L3G_INT1_DURATION 0x38

9.15.1 Define Documentation

9.15.1.1 #define L3G_CTRL_REG1 0x20

Definition at line 8 of file L3G.h.

9.15.1.2 #define L3G_CTRL_REG2 0x21

Definition at line 9 of file L3G.h.

9.15.1.3 #define L3G_CTRL_REG3 0x22

Definition at line 10 of file L3G.h.

9.15.1.4 #define L3G_CTRL_REG4 0x23

Definition at line 11 of file L3G.h.

9.15.1.5 #define L3G_CTRL_REG5 0x24

Definition at line 12 of file L3G.h.

9.15.1.6 #define L3G_FIFO_CTRL_REG 0x2E

Definition at line 24 of file L3G.h.

9.15.1.7 #define L3G_FIFO_SRC_REG 0x2F

Definition at line 25 of file L3G.h.

9.15.1.8 #define L3G_INT1_CFG 0x30

Definition at line 27 of file L3G.h.

9.15.1.9 #define L3G_INT1_DURATION 0x38

Definition at line 35 of file L3G.h.

9.15.1.10 #define L3G_INT1_SRC 0x31

Definition at line 28 of file L3G.h.

9.15.1.11 #define L3G_INT1_THS_XH 0x32

Definition at line 29 of file L3G.h.

9.15.1.12 #define L3G_INT1_THS_XL 0x33

Definition at line 30 of file L3G.h.

9.15.1.13 #define L3G_INT1_THS_YH 0x34

Definition at line 31 of file L3G.h.

9.15.1.14 #define L3G_INT1_THS_YL 0x35

Definition at line 32 of file L3G.h.

9.15.1.15 #define L3G_INT1_THS_ZH 0x36

Definition at line 33 of file L3G.h.

9.15.1.16 #define L3G_INT1_THS_ZL 0x37

Definition at line 34 of file L3G.h.

9.15.1.17 #define L3G_OUT_TEMP 0x26

Definition at line 14 of file L3G.h.

9.15.1.18 #define L3G_OUT_X_H 0x29

Definition at line 18 of file L3G.h.

9.15.1.19 #define L3G_OUT_X_L 0x28

Definition at line 17 of file L3G.h.

9.15.1.20 #define L3G_OUT_Y_H 0x2B

Definition at line 20 of file L3G.h.

9.15.1.21 #define L3G_OUT_Y_L 0x2A

Definition at line 19 of file L3G.h.

9.15.1.22 #define L3G_OUT_Z_H 0x2D

Definition at line 22 of file L3G.h.

9.15.1.23 #define L3G_OUT_Z_L 0x2C

Definition at line 21 of file L3G.h.

9.15.1.24 #define L3G_REFERENCE 0x25

Definition at line 13 of file L3G.h.

9.15.1.25 #define L3G_STATUS_REG 0x27

Definition at line 15 of file L3G.h.

9.15.1.26 #define L3G_WHO_AM_I 0x0F

Definition at line 6 of file L3G.h.

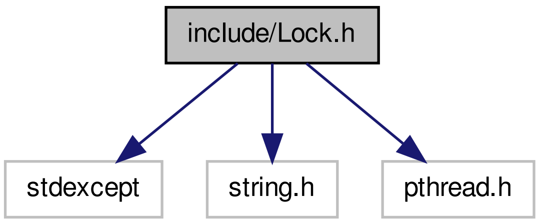
9.16 include/L3G.h

```
00001 #ifndef _L3G_h
00002 #define _L3G_h
00003
00004 #include "I2CBus.h"
00005
00006 #define L3G_WHO_AM_I      0x0F
00007
00008 #define L3G_CTRL_REG1     0x20
00009 #define L3G_CTRL_REG2     0x21
00010 #define L3G_CTRL_REG3     0x22
00011 #define L3G_CTRL_REG4     0x23
00012 #define L3G_CTRL_REG5     0x24
00013 #define L3G_REFERENCE     0x25
00014 #define L3G_OUT_TEMP      0x26
00015 #define L3G_STATUS_REG    0x27
00016
00017 #define L3G_OUT_X_L       0x28
00018 #define L3G_OUT_X_H       0x29
00019 #define L3G_OUT_Y_L       0x2A
00020 #define L3G_OUT_Y_H       0x2B
00021 #define L3G_OUT_Z_L       0x2C
00022 #define L3G_OUT_Z_H       0x2D
00023
00024 #define L3G_FIFO_CTRL_REG 0x2E
00025 #define L3G_FIFO_SRC_REG  0x2F
00026
00027 #define L3G_INT1_CFG      0x30
00028 #define L3G_INT1_SRC      0x31
00029 #define L3G_INT1_THS_XH   0x32
00030 #define L3G_INT1_THS_XL   0x33
00031 #define L3G_INT1_THS_YH   0x34
00032 #define L3G_INT1_THS_YL   0x35
00033 #define L3G_INT1_THS_ZH   0x36
00034 #define L3G_INT1_THS_ZL   0x37
00035 #define L3G_INT1_DURATION 0x38
00036
00037
00044 class L3G
00045 {
00046 public:
00052     L3G(const char * i2cDeviceName);
00053
00054     int g[3];
00060     void enable(void);
00061
00070     void writeReg(uint8_t reg, uint8_t value);
00071
00078     uint8_t readReg(uint8_t reg);
```

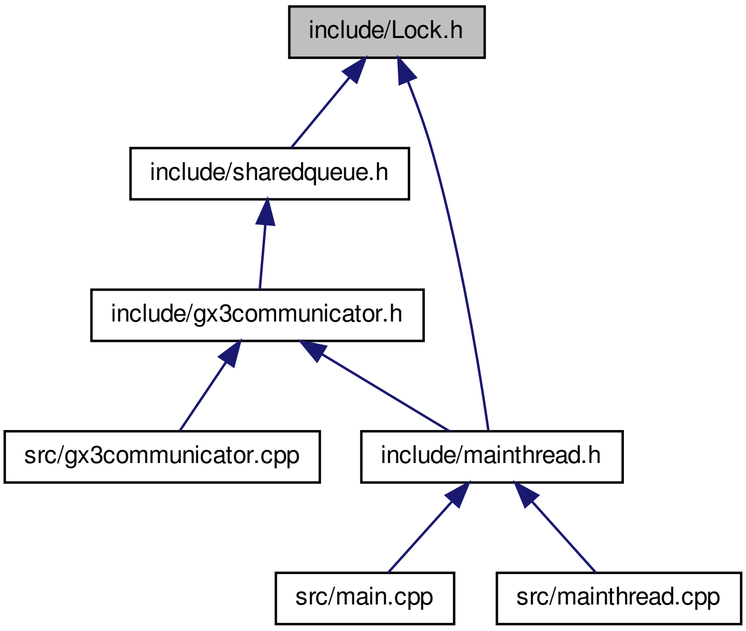
```
    void read();

private:
    void detectAddress();
    I2CBus i2c;
};

#endif
```

9.17 include/Lock.h File Reference

`#include <stdexcept>` `#include <string.h>` `#include <pthread.h>` Include dependency graph for Lock.h:



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



Classes

- class USU::Lock

  Wrapper class for pthread mutexes.
- class USU::ScopedLock

  Provides a helper class for Scoped Mutexes.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

9.17.1 Detailed Description

Small C++ wrapper classes for pthread mutexes

Author

Jan Sommer Created on: Apr 10, 2013

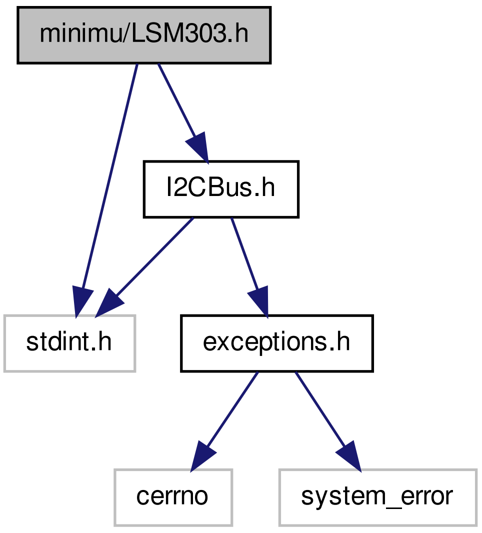
Definition in file Lock.h.

9.18 include/Lock.h

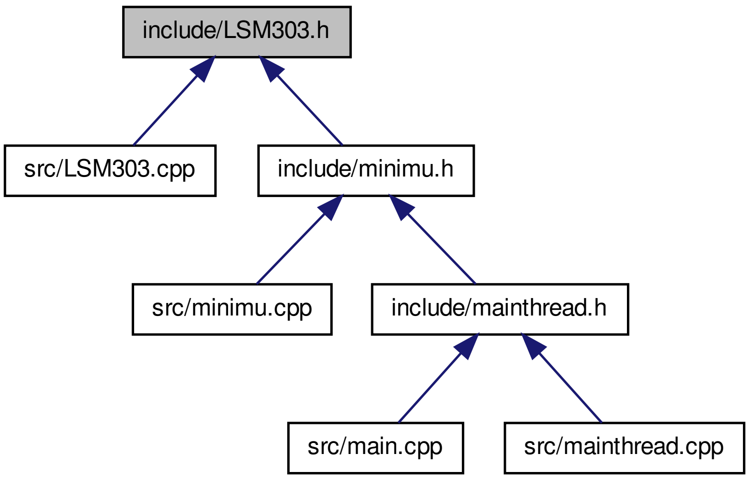
```

#ifndef LOCK_H
#define LOCK_H

#include <stdexcept>
#include <string.h>
#include <pthread.h>

namespace USU {

class Lock
{
private:
    pthread_mutex_t mMutex;
    Lock(const Lock& arg);
    Lock& operator=(const Lock& rhs);
public:

    Lock();
    virtual ~Lock();
    void lock();
    void unlock();
};

inline
Lock::Lock()
{
    int ret;
    if ( (ret = pthread_mutex_init(&mMutex, NULL)) != 0)
    {
        throw std::runtime_error(strerror(errno));
    }
}

inline
Lock::~Lock()
{
    int ret;
    if ( (ret = pthread_mutex_destroy(&mMutex) ) != 0)
    {
        throw std::runtime_error(strerror(errno));
    }
}


inline
void Lock::lock()
{
    pthread_mutex_lock(&mMutex);
}

inline
void Lock::unlock()
{
    pthread_mutex_unlock(&mMutex);
}


class ScopedLock
{
private:
    Lock &mLock;
    ScopedLock(const ScopedLock& thread);
    ScopedLock& operator=(const ScopedLock& rhs);
public:
    ScopedLock(Lock &lock);

    virtual ~ScopedLock();
};

inline
ScopedLock::ScopedLock(Lock &lock)
    :mLock(lock)
{
    mLock.lock();
}

inline
ScopedLock::~ScopedLock()
{
    mLock.unlock();
}

}

#endif // LOCK_H
```

9.19 include/LSM303.h File Reference

`#include <stdint.h>` `#include "I2CBus.h"` Include dependency graph for LSM303.h:



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



Classes

- class LSM303

  Class to manage communication to the LSM303 compass via the I2C-bus.

Defines

- #define LSM303_CTRL_REG1_A 0x20
- #define LSM303_CTRL_REG2_A 0x21
- #define LSM303_CTRL_REG3_A 0x22
- #define LSM303_CTRL_REG4_A 0x23
- #define LSM303_CTRL_REG5_A 0x24
- #define LSM303_CTRL_REG6_A 0x25
- #define LSM303_HP_FILTER_RESET_A 0x25
- #define LSM303_REFERENCE_A 0x26
- #define LSM303_STATUS_REG_A 0x27
- #define LSM303_OUT_X_L_A 0x28
- #define LSM303_OUT_X_H_A 0x29
- #define LSM303_OUT_Y_L_A 0x2A
- #define LSM303_OUT_Y_H_A 0x2B
- #define LSM303_OUT_Z_L_A 0x2C
- #define LSM303_OUT_Z_H_A 0x2D

- #define LSM303_FIFO_CTRL_REG_A 0x2E
- #define LSM303_FIFO_SRC_REG_A 0x2F
- #define LSM303_INT1_CFG_A 0x30
- #define LSM303_INT1_SRC_A 0x31
- #define LSM303_INT1_THS_A 0x32
- #define LSM303_INT1_DURATION_A 0x33
- #define LSM303_INT2_CFG_A 0x34
- #define LSM303_INT2_SRC_A 0x35
- #define LSM303_INT2_THS_A 0x36
- #define LSM303_INT2_DURATION_A 0x37
- #define LSM303_CLICK_CFG_A 0x38
- #define LSM303_CLICK_SRC_A 0x39
- #define LSM303_CLICK_THS_A 0x3A
- #define LSM303_TIME_LIMIT_A 0x3B
- #define LSM303_TIME_LATENCY_A 0x3C
- #define LSM303_TIME_WINDOW_A 0x3D
- #define LSM303_CRA_REG_M 0x00
- #define LSM303_CRB_REG_M 0x01
- #define LSM303_MR_REG_M 0x02
- #define LSM303_OUT_X_H_M 0x03
- #define LSM303_OUT_X_L_M 0x04
- #define LSM303_OUT_Y_H_M -1
- #define LSM303_OUT_Y_L_M -2
- #define LSM303_OUT_Z_H_M -3
- #define LSM303_OUT_Z_L_M -4
- #define LSM303_SR_REG_M 0x09
- #define LSM303_IRA_REG_M 0x0A
- #define LSM303_IRB_REG_M 0x0B
- #define LSM303_IRC_REG_M 0x0C
- #define LSM303_WHO_AM_I_M 0x0F
- #define LSM303_TEMP_OUT_H_M 0x31
- #define LSM303_TEMP_OUT_L_M 0x32
- #define LSM303DLH_OUT_Y_H_M 0x05
- #define LSM303DLH_OUT_Y_L_M 0x06
- #define LSM303DLH_OUT_Z_H_M 0x07
- #define LSM303DLH_OUT_Z_L_M 0x08
- #define LSM303DLM_OUT_Z_H_M 0x05
- #define LSM303DLM_OUT_Z_L_M 0x06
- #define LSM303DLM_OUT_Y_H_M 0x07
- #define LSM303DLM_OUT_Y_L_M 0x08
- #define LSM303DLHC_OUT_Z_H_M 0x05
- #define LSM303DLHC_OUT_Z_L_M 0x06

9.19.1 Define Documentation

9.19.1.1 #define **LSM303_CLICK_CFG_A** 0x38

Definition at line 38 of file LSM303.h.

9.19.1.2 #define LSM303_CLICK_SRC_A 0x39

Definition at line 39 of file LSM303.h.

9.19.1.3 #define LSM303_CLICK_THS_A 0x3A

Definition at line 40 of file LSM303.h.

9.19.1.4 #define LSM303_CRA_REG_M 0x00

Definition at line 45 of file LSM303.h.

9.19.1.5 #define LSM303_CRB_REG_M 0x01

Definition at line 46 of file LSM303.h.

9.19.1.6 #define LSM303_CTRL_REG1_A 0x20

Definition at line 9 of file LSM303.h.

9.19.1.7 #define LSM303_CTRL_REG2_A 0x21

Definition at line 10 of file LSM303.h.

9.19.1.8 #define LSM303_CTRL_REG3_A 0x22

Definition at line 11 of file LSM303.h.

9.19.1.9 #define LSM303_CTRL_REG4_A 0x23

Definition at line 12 of file LSM303.h.

9.19.1.10 #define LSM303_CTRL_REG5_A 0x24

Definition at line 13 of file LSM303.h.

9.19.1.11 #define LSM303_CTRL_REG6_A 0x25

Definition at line 14 of file LSM303.h.

9.19.1.12 #define LSM303_FIFO_CTRL_REG_A 0x2E

Definition at line 26 of file LSM303.h.

9.19.1.13 #define LSM303_FIFO_SRC_REG_A 0x2F

Definition at line 27 of file LSM303.h.

9.19.1.14 #define LSM303_HP_FILTER_RESET_A 0x25

Definition at line 15 of file LSM303.h.

9.19.1.15 #define **LSM303_INT1_CFG_A** 0x30

Definition at line 29 of file LSM303.h.

9.19.1.16 #define **LSM303_INT1_DURATION_A** 0x33

Definition at line 32 of file LSM303.h.

9.19.1.17 #define **LSM303_INT1_SRC_A** 0x31

Definition at line 30 of file LSM303.h.

9.19.1.18 #define **LSM303_INT1_THS_A** 0x32

Definition at line 31 of file LSM303.h.

9.19.1.19 #define **LSM303_INT2_CFG_A** 0x34

Definition at line 33 of file LSM303.h.

9.19.1.20 #define **LSM303_INT2_DURATION_A** 0x37

Definition at line 36 of file LSM303.h.

9.19.1.21 #define **LSM303_INT2_SRC_A** 0x35

Definition at line 34 of file LSM303.h.

9.19.1.22 #define **LSM303_INT2_THS_A** 0x36

Definition at line 35 of file LSM303.h.

9.19.1.23 #define **LSM303_IRA_REG_M** 0x0A

Definition at line 57 of file LSM303.h.

9.19.1.24 #define **LSM303_IRB_REG_M** 0x0B

Definition at line 58 of file LSM303.h.

9.19.1.25 #define **LSM303_IRC_REG_M** 0x0C

Definition at line 59 of file LSM303.h.

9.19.1.26 #define **LSM303_MR_REG_M** 0x02

Definition at line 47 of file LSM303.h.

9.19.1.27 #define **LSM303_OUT_X_H_A** 0x29

Definition at line 20 of file LSM303.h.

9.19.1.28 #define LSM303_OUT_X_H_M 0x03

Definition at line 49 of file LSM303.h.

9.19.1.29 #define LSM303_OUT_X_L_A 0x28

Definition at line 19 of file LSM303.h.

9.19.1.30 #define LSM303_OUT_X_L_M 0x04

Definition at line 50 of file LSM303.h.

9.19.1.31 #define LSM303_OUT_Y_H_A 0x2B

Definition at line 22 of file LSM303.h.

9.19.1.32 #define LSM303_OUT_Y_H_M -1

Definition at line 51 of file LSM303.h.

9.19.1.33 #define LSM303_OUT_Y_L_A 0x2A

Definition at line 21 of file LSM303.h.

9.19.1.34 #define LSM303_OUT_Y_L_M -2

Definition at line 52 of file LSM303.h.

9.19.1.35 #define LSM303_OUT_Z_H_A 0x2D

Definition at line 24 of file LSM303.h.

9.19.1.36 #define LSM303_OUT_Z_H_M -3

Definition at line 53 of file LSM303.h.

9.19.1.37 #define LSM303_OUT_Z_L_A 0x2C

Definition at line 23 of file LSM303.h.

9.19.1.38 #define LSM303_OUT_Z_L_M -4

Definition at line 54 of file LSM303.h.

9.19.1.39 #define LSM303_REFERENCE_A 0x26

Definition at line 16 of file LSM303.h.

9.19.1.40 #define LSM303_SR_REG_M 0x09

Definition at line 56 of file LSM303.h.

9.19.1.41 #define LSM303_STATUS_REG_A 0x27

Definition at line 17 of file LSM303.h.

9.19.1.42 #define LSM303_TEMP_OUT_H_M 0x31

Definition at line 63 of file LSM303.h.

9.19.1.43 #define LSM303_TEMP_OUT_L_M 0x32

Definition at line 64 of file LSM303.h.

9.19.1.44 #define LSM303_TIME_LATENCY_A 0x3C

Definition at line 42 of file LSM303.h.

9.19.1.45 #define LSM303_TIME_LIMIT_A 0x3B

Definition at line 41 of file LSM303.h.

9.19.1.46 #define LSM303_TIME_WINDOW_A 0x3D

Definition at line 43 of file LSM303.h.

9.19.1.47 #define LSM303_WHO_AM_I_M 0x0F

Definition at line 61 of file LSM303.h.

9.19.1.48 #define LSM303DLH_OUT_Y_H_M 0x05

Definition at line 65 of file LSM303.h.

9.19.1.49 #define LSM303DLH_OUT_Y_L_M 0x06

Definition at line 66 of file LSM303.h.

9.19.1.50 #define LSM303DLH_OUT_Z_H_M 0x07

Definition at line 67 of file LSM303.h.

9.19.1.51 #define LSM303DLH_OUT_Z_L_M 0x08

Definition at line 68 of file LSM303.h.

9.19.1.52 #define LSM303DLHC_OUT_Z_H_M 0x05

Definition at line 75 of file LSM303.h.

9.19.1.53 #define LSM303DLHC_OUT_Z_L_M 0x06

Definition at line 76 of file LSM303.h.

9.19.1.54 #define LSM303DLM_OUT_Y_H_M 0x07

Definition at line 72 of file LSM303.h.

9.19.1.55 #define LSM303DLM_OUT_Y_L_M 0x08

Definition at line 73 of file LSM303.h.

9.19.1.56 #define LSM303DLM_OUT_Z_H_M 0x05

Definition at line 70 of file LSM303.h.

9.19.1.57 #define LSM303DLM_OUT_Z_L_M 0x06

Definition at line 71 of file LSM303.h.

9.20 include/LSM303.h

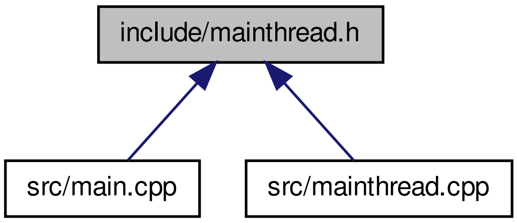
```
#ifndef LSM303_h
#define LSM303_h

#include <stdint.h>
#include "I2CBus.h"

// register addresses

#define LSM303_CTRL_REG1_A       0x20
#define LSM303_CTRL_REG2_A       0x21
#define LSM303_CTRL_REG3_A       0x22
#define LSM303_CTRL_REG4_A       0x23
#define LSM303_CTRL_REG5_A       0x24
#define LSM303_CTRL_REG6_A       0x25 // DLHC only
#define LSM303_HP_FILTER_RESET_A 0x25 // DLH, DLM only
#define LSM303_REFERENCE_A       0x26
#define LSM303_STATUS_REG_A      0x27

#define LSM303_OUT_X_L_A         0x28
#define LSM303_OUT_X_H_A         0x29
#define LSM303_OUT_Y_L_A         0x2A
#define LSM303_OUT_Y_H_A         0x2B
#define LSM303_OUT_Z_L_A         0x2C
#define LSM303_OUT_Z_H_A         0x2D

#define LSM303_FIFO_CTRL_REG_A   0x2E // DLHC only
#define LSM303_FIFO_SRC_REG_A    0x2F // DLHC only

#define LSM303_INT1_CFG_A        0x30
#define LSM303_INT1_SRC_A        0x31
#define LSM303_INT1_THS_A        0x32
#define LSM303_INT1_DURATION_A   0x33
#define LSM303_INT2_CFG_A        0x34
#define LSM303_INT2_SRC_A        0x35
#define LSM303_INT2_THS_A        0x36
#define LSM303_INT2_DURATION_A   0x37

#define LSM303_CLICK_CFG_A       0x38 // DLHC only
#define LSM303_CLICK_SRC_A       0x39 // DLHC only
#define LSM303_CLICK_THS_A       0x3A // DLHC only
#define LSM303_TIME_LIMIT_A      0x3B // DLHC only
#define LSM303_TIME_LATENCY_A    0x3C // DLHC only
#define LSM303_TIME_WINDOW_A     0x3D // DLHC only

#define LSM303_CRA_REG_M         0x00
#define LSM303_CRB_REG_M         0x01
#define LSM303_MR_REG_M          0x02

```

```
#define LSM303_OUT_X_H_M         0x03
#define LSM303_OUT_X_L_M         0x04
#define LSM303_OUT_Y_H_M         -1   // The addresses of the Y and Z
 magnetometer output registers
#define LSM303_OUT_Y_L_M         -2   // are reversed on the DLM and DLHC
 relative to the DLH.
#define LSM303_OUT_Z_H_M         -3   // These four defines have dummy values
 so the library can
#define LSM303_OUT_Z_L_M         -4   // determine the correct address based on
 the device type.

#define LSM303_SR_REG_M          0x09
#define LSM303_IRA_REG_M         0x0A
#define LSM303_IRB_REG_M         0x0B
#define LSM303_IRC_REG_M         0x0C

#define LSM303_WHO_AM_I_M        0x0F // DLM only

#define LSM303_TEMP_OUT_H_M      0x31 // DLHC only
#define LSM303_TEMP_OUT_L_M      0x32 // DLHC only
#define LSM303DLH_OUT_Y_H_M      0x05
#define LSM303DLH_OUT_Y_L_M      0x06
#define LSM303DLH_OUT_Z_H_M      0x07
#define LSM303DLH_OUT_Z_L_M      0x08

#define LSM303DLM_OUT_Z_H_M      0x05
#define LSM303DLM_OUT_Z_L_M      0x06
#define LSM303DLM_OUT_Y_H_M      0x07
#define LSM303DLM_OUT_Y_L_M      0x08

#define LSM303DLHC_OUT_Z_H_M     0x05
#define LSM303DLHC_OUT_Z_L_M     0x06

class LSM303
{
 public:
    int a[3];
    int m[3];
    LSM303(const char * i2cDeviceName);

    void enable(void);

    void writeAccReg(uint8_t reg, uint8_t value);

    uint8_t readAccReg(uint8_t reg);

    void writeMagReg(uint8_t reg, uint8_t value);

    uint8_t readMagReg(uint8_t reg);

    void readAcc(void);

    void readMag(void);

    void read(void);

private:
    I2CBus i2c_mag, i2c_acc;
    enum class Device {
        LSM303DLH,
        LSM303DLM,
        LSM303DLHC,
    } device;
};

#endif
```

9.21 include/mainthread.h File Reference

`#include "periodicrtthread.h" #include "minimu.h" #include "Lock.h" #include "gx3communicator.h" #include "messages.-h" #include "motorcontrol.h"` Include dependency graph for mainthread.h:

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



Classes

- class USU::MainThread

  Represents the Periodic Thread class for state estimation.
- struct **USU::MainThread::Command**

  Struct representing a single command point.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

9.21.1 Detailed Description

C++ class for the sensor fusion and state estimated. Based on the PeriodicRtThread class.

Author

Jan Sommer Created on: Apr 20, 2013

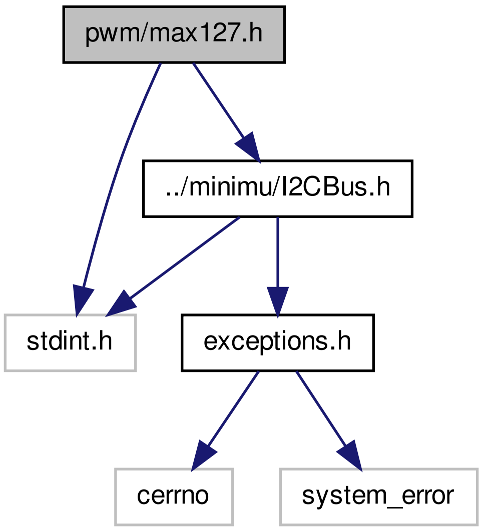
Definition in file mainthread.h.

9.22 include/mainthread.h

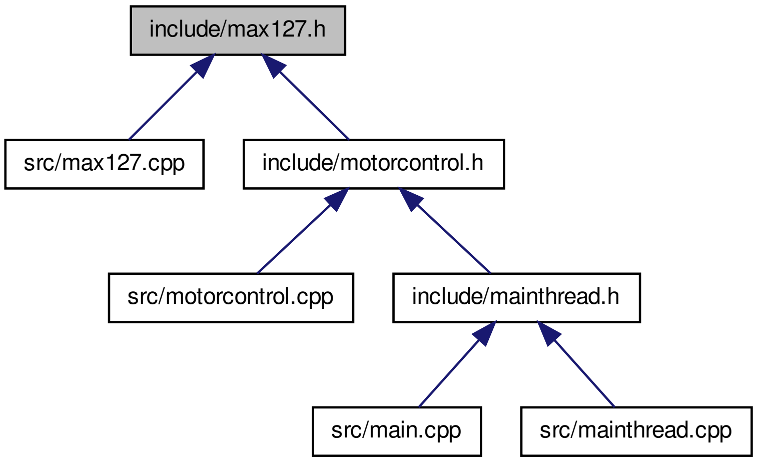
```

#ifndef MAINTHREAD_H
#define MAINTHREAD_H

#include "periodicrtthread.h"
#include "minimu.h"
#include "Lock.h"
#include "gx3communicator.h"
#include "messages.h"
#include "motorcontrol.h"

namespace USU
{

class MainThread : public PeriodicRtThread
{
public:
    enum Mode
    {
        SimpleControl,
        CollectPololuData,
        CollectMicroStrainData,
        CollectData
    };

    MainThread(int priority, unsigned int period_us, const char* i2cImu, const
char *i2cMotor);

    virtual void run();

    void stop() { mKeepRunning = false; }

    bool getState();

    void initializeModeSimpleControl(std::string trajFilename, float pgain);

    Mode getMode() const;
    void setMode(const Mode &value);

private:
    void runSimpleControl();

    void runCollectPololu();

    void runCollectMicroStrain();

    void runCollectBoth();

    Mode mMode;

    struct Command
    {
        unsigned int time;
        Eigen::Vector3f angVel;
    };
    std::vector<Command> mCommandList;
    MinImu mImu;
    GX3Communicator mGX3;
    MotorControl mMotors;

    bool mState;
    Lock mStateLock;
    volatile bool mKeepRunning;
    MainThread(const MainThread& thread);
    MainThread& operator=(const MainThread& rhs);
};

}

#endif // MAINTHREAD_H
```

9.23 include/max127.h File Reference

`#include <stdint.h>` `#include "I2CBus.h"` Include dependency graph for max127.h:



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



Classes

- class USU::Max127

  Class representing the MAX127 ADC.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

Variables

- const uint8_t USU::I2C_ADDRESS = 0b00101000

  I2C-address of the ADC.
- const uint8_t USU::CONTROL_BYTE = 0b10000110

  Template of the control byte.
- const uint8_t USU::SEL0 = 4

9.23.1 Detailed Description

C++ class for the ADC Max127.

Author

Jan Sommer Created on: May 20, 2013

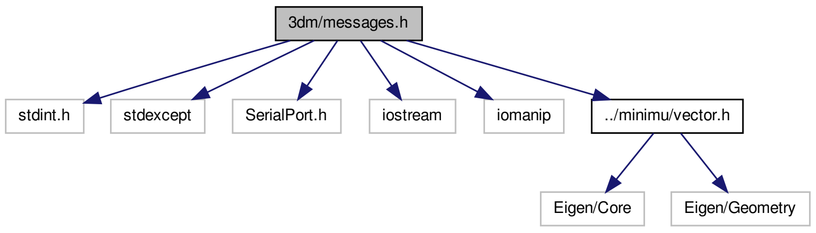
Definition in file max127.h.

9.24 include/max127.h

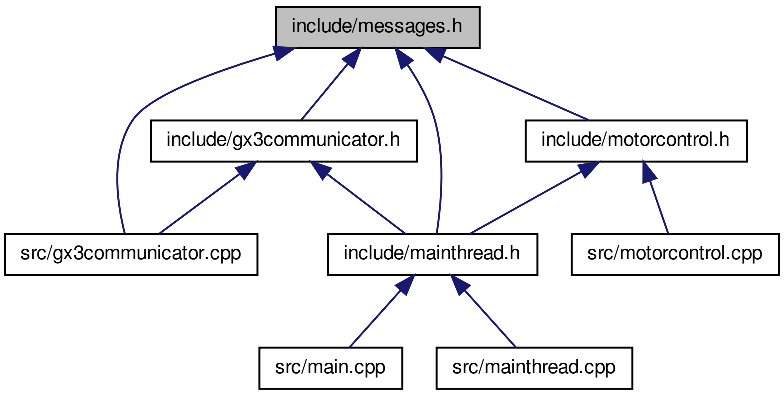
```
#ifndef MAX127_H
#define MAX127_H

#include <stdint.h>

#include "I2CBus.h"

namespace USU
{

const uint8_t I2C_ADDRESS  = 0b00101000;

const uint8_t CONTROL_BYTE = 0b10000110;
const uint8_t SEL0         = 4;
class Max127
{
public:
    Max127(const char *i2cdevice);

    int16_t readRaw(uint8_t channel);
    float readVoltage(unsigned int channel);

private:
    I2CBus mI2c;
};

}

#endif // MAX127_H
```

9.25 include/messages.h File Reference

`#include <stdint.h>` `#include <stdexcept>` `#include <SerialPort.h>` `#include <iostream>` `#include <iomanip>` `#include "vector.h"` Include dependency graph for messages.h:



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



Classes

- class USU::GX3Packet

  Abstract base class for received packets.
- class USU::RawAccAng

  Representation for receiving (raw) acceleration & angular rate packets.
- class USU::AccAngMag

  Representation for receiving acceleration, angular rate and magnetometer packets.
- class USU::Quaternion

  Representation for receiving the Quaternion representation from the IMU.
- class USU::AccAngMagOrientationMat

  Representation for packets containing the 3 sensor vectors and orientation matrix This class can be used with the commands which return 3 Vectors and a 3x3 Matrix. The units are:
- class USU::GX3Command

  Base class for commands send to the 3DM-GX3-25.
- class USU::SetCountinuousMode

  Represents the "Set continuous mode" command.
- class USU::SamplingSettings

  Represents the "Sampling Settings" command.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

Variables

- const uint8_t USU::RAW_ACC_ANG = 0xC1
- const uint8_t USU::ACC_ANG = 0xC2
- const uint8_t USU::DELTA_ANGLE_VEL = 0xC3
- const uint8_t USU::SET_CONTINUOUS_MODE = 0xC4
- const uint8_t USU::ORIENTATION_MATRIX = 0xC5
- const uint8_t USU::ORIENTATION_UPDATE_MAT = 0xC6
- const uint8_t USU::MAG_VEC = 0xC7
- const uint8_t USU::ACC_ANG_ORIENTATION_MAT = 0xC8
- const uint8_t USU::WRITE_ACC_BIAS_CORRECTION = 0xC9
- const uint8_t USU::WRITE_GYRO_BIAS_CORRECTION = 0xCA
- const uint8_t USU::ACC_ANG_MAG_VEC = 0xCB
- const uint8_t USU::ACC_ANG_MAG_VEC_ORIENTATION_MAT = 0xCC
- const uint8_t USU::CAPTURE_GYRO_BIAS = 0xCD
- const uint8_t USU::EULER_ANGLES = 0xCE
- const uint8_t USU::EULER_ANGLES_ANG_RATES = 0xCF
- const uint8_t USU::TRANSFER_TO_NONVOL_MEM = 0xD0
- const uint8_t USU::TEMPERATURES = 0xD1
- const uint8_t USU::GYRO_STABIL_ACC_ANG_MAG = 0xD2
- const uint8_t USU::DELTA_ANGLE_VEL_MAG_VEC = 0xD3
- const uint8_t USU::MODE = 0xD4
- const uint8_t USU::MODE_PRESET = 0xD5
- const uint8_t USU::CONTINUOUS_PRESET = 0xD6
- const uint8_t USU::TIMER = 0xD7
- const uint8_t USU::COMM_SETTINGS = 0xD9
- const uint8_t USU::STATIONARY_TEST = 0xDA
- const uint8_t USU::SAMPLING_SETTINGS = 0xDB
- const uint8_t USU::REALIGN_UP_NORTH = 0xDD
- const uint8_t USU::QUATERNION = 0xDF
- const uint8_t USU::WRITE_WORD_EEPROM = 0xE4
- const uint8_t USU::READ_WORD_EEPROM = 0xE5
- const uint8_t USU::READ_FIRMWARE_VER = 0xE9
- const uint8_t USU::READ_DEVICE_ID = 0xEA
- const uint8_t USU::STOP_CONTINUOUS = 0xFA
- const uint8_t USU::FIRMWARE_UPDATE = 0xFD
- const uint8_t USU::DEVICE_RESET = 0xFE

9.25.1 Detailed Description

File containing classes representing messages of the single byte protocol for the 3DM-GX3-25

Author

Jan Sommer Created on: Apr 25, 2013

Definition in file messages.h.

9.26 include/messages.h

```
#ifndef MESSAGES_H
#define MESSAGES_H

#include<stdint.h>
#include<stdexcept>
#include<SerialPort.h>
#include<iostream>
#include<iomanip>

#include "vector.h"

namespace USU
{

// Command protocol constants

const uint8_t RAW_ACC_ANG                          = 0xC1;
const uint8_t ACC_ANG                              = 0xC2;
const uint8_t DELTA_ANGLE_VEL                      = 0xC3;
const uint8_t SET_CONTINUOUS_MODE                  = 0xC4;
const uint8_t ORIENTATION_MATRIX                   = 0xC5;
const uint8_t ORIENTATION_UPDATE_MAT               = 0xC6;
const uint8_t MAG_VEC                              = 0xC7;
const uint8_t ACC_ANG_ORIENTATION_MAT              = 0xC8;
const uint8_t WRITE_ACC_BIAS_CORRECTION            = 0xC9;
const uint8_t WRITE_GYRO_BIAS_CORRECTION           = 0xCA;
const uint8_t ACC_ANG_MAG_VEC                      = 0xCB;
const uint8_t ACC_ANG_MAG_VEC_ORIENTATION_MAT = 0xCC;
const uint8_t CAPTURE_GYRO_BIAS                    = 0xCD;
const uint8_t EULER_ANGLES                         = 0xCE;
const uint8_t EULER_ANGLES_ANG_RATES               = 0xCF;
const uint8_t TRANSFER_TO_NONVOL_MEM               = 0xD0;
const uint8_t TEMPERATURES                         = 0xD1;
const uint8_t GYRO_STABIL_ACC_ANG_MAG              = 0xD2;
const uint8_t DELTA_ANGLE_VEL_MAG_VEC              = 0xD3;
const uint8_t MODE                                 = 0xD4;
const uint8_t MODE_PRESET                          = 0xD5;
const uint8_t CONTINUOUS_PRESET                    = 0xD6;
const uint8_t TIMER                                = 0xD7;
const uint8_t COMM_SETTINGS                        = 0xD9;
const uint8_t STATIONARY_TEST                      = 0xDA;
const uint8_t SAMPLING_SETTINGS                    = 0xDB;
const uint8_t REALIGN_UP_NORTH                     = 0xDD;
const uint8_t QUATERNION                           = 0xDF;
const uint8_t WRITE_WORD_EEPROM                    = 0xE4;
const uint8_t READ_WORD_EEPROM                     = 0xE5;
const uint8_t READ_FIRMWARE_VER                    = 0xE9;
const uint8_t READ_DEVICE_ID                       = 0xEA;
const uint8_t STOP_CONTINUOUS                      = 0xFA;
const uint8_t FIRMWARE_UPDATE                      = 0xFD;
const uint8_t DEVICE_RESET                         = 0xFE;
class GX3Packet
{
public:
     virtual bool readFromSerial(SerialPort &serialPort) = 0;

     virtual void print(std::ostream &os) const = 0;

    static bool calculateChecksum(uint8_t * buffer, unsigned int length)
    {
        uint16_t sum = 0;
        for(unsigned int i = 0; i<length-2; i++)
        {
            sum += buffer[i];
        }
        uint16_t temp = (buffer[length-2] << 8)  + buffer[length-1];
        return (sum == temp );
    }


```

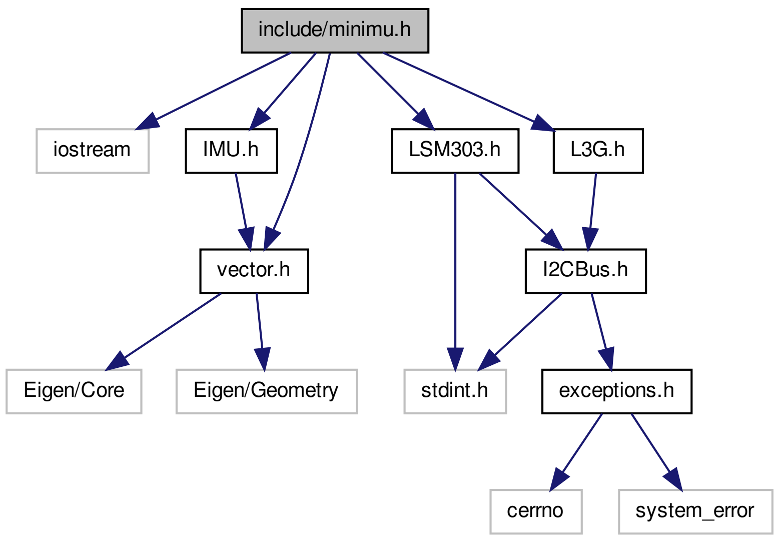
```cpp
protected:
    static inline vector createVector(uint8_t * buffer)
    {
        return vector(*(float*) &buffer[0],
                      *(float*) &buffer[4],
                      *(float*) &buffer[8]);
    }

    static inline unsigned int createUInt(uint8_t *buffer)
    {
        return ( (buffer[0]<<24) + (buffer[1]<<16) + (buffer[2]<<8) + buffer[3] );
    }

    static void createMatrix(uint8_t * buffer, matrix& mat)
    {
        mat << *(float*) &buffer[0],  *(float*) &buffer[4],  *(float*) &buffer[8],
               *(float*) &buffer[12], *(float*) &buffer[16], *(float*) &buffer[20],
               *(float*) &buffer[24], *(float*) &buffer[28], *(float*) &buffer[32];
    }
};

static std::ostream & operator << (std::ostream & os, const GX3Packet & packet)
{
   packet.print(os);
   return os;
}



class RawAccAng : public GX3Packet
{
public:
    RawAccAng() {}

    bool readFromSerial(SerialPort &serialPort)
    {
        uint8_t buffer[size];
        buffer[0] = serialPort.ReadByte();
        if(buffer[0] != RAW_ACC_ANG && buffer[0] != ACC_ANG) return false;

        serialPort.ReadRaw(&buffer[1], size-1);
        if(GX3Packet::calculateChecksum(buffer, size) == false)
        {
            using namespace std;
            return false;
        }

        acc  = createVector(&buffer[1]);
        gyro = createVector(&buffer[13]);

        timer = createUInt(&buffer[25]);

        return true;
    }

    virtual void print(std::ostream &os) const
    {
        os << timer/63 << ",\t" << acc(0)  << ", " << acc(1) << ", "  << acc(2)
                    << ",\t" << gyro(0) << ", " << gyro(1) << ", " << gyro(2);
    }

    vector acc;
    vector gyro;
    unsigned int timer;
    enum{size = 31};
};

class AccAngMag : public GX3Packet
{
public:
    AccAngMag() {}

    bool readFromSerial(SerialPort &serialPort)
    {
        unsigned count = 10;
        uint8_t buffer[size];
        do
        {
            buffer[0] = serialPort.ReadByte();
        } while(--count && buffer[0] != ACC_ANG_MAG_VEC);

        if(count == 0)
        {
            return false; //throw std::runtime_error("Wrong package
 identifier");
        }

        serialPort.ReadRaw(&buffer[1], size-1);
        if(GX3Packet::calculateChecksum(buffer, size) == false)
        {
            using namespace std;
            return false;
        }

        acc  = createVector(&buffer[1]);
        gyro = createVector(&buffer[13]);
        mag  = createVector(&buffer[25]);

        timer = createUInt(&buffer[37]);

        return true;
    }

    virtual void print(std::ostream &os) const
    {
        os << timer/63 << ",\t" << acc(0)  << ", " << acc(1)  << ", " << acc(2)
                    << ",\t" << mag(0)  << ", " << mag(1)  << ", " << mag(2)
                    << ",\t" << gyro(0) << ", " << gyro(1) << ", " << gyro(2);
    }

    vector acc;
    vector gyro;
    vector mag;
    unsigned int timer;
    enum{size = 43};
};

class Quaternion : public GX3Packet
{
public:
    Quaternion() {}

    bool readFromSerial(SerialPort &serialPort)
    {
        uint8_t buffer[size];
        buffer[0] = serialPort.ReadByte();
        if(buffer[0] != QUATERNION)
        {
            return false; //throw std::runtime_error("Wrong package
 identifier");
        }
        serialPort.ReadRaw(&buffer[1], size-1);
        if(GX3Packet::calculateChecksum(buffer, size) == false)
        {
            return false;
        }

        quat = quaternion( (float*) &buffer[1]);
        timer = createUInt(&buffer[17]);

        return true;
    }

    virtual void print(std::ostream &os) const
    {
        os << timer << ",\t" << quat.w()  << ", " << quat.x() << ", " << quat.y
() << ", " << quat.z();
    }

    quaternion quat;
    unsigned int timer;
    enum{size = 23};
};


class AccAngMagOrientationMat : public GX3Packet
{
public:
    AccAngMagOrientationMat() {}

    bool readFromSerial(SerialPort &serialPort)
    {
        uint8_t buffer[size];
        buffer[0] = serialPort.ReadByte();
        if(buffer[0] != ACC_ANG_MAG_VEC_ORIENTATION_MAT) return false;

        serialPort.ReadRaw(&buffer[1], size-1);
        if(GX3Packet::calculateChecksum(buffer, size) == false) return false;

        acc  = createVector(&buffer[1]);
        gyro = createVector(&buffer[13]);
        mag  = createVector(&buffer[25]);

        createMatrix(&buffer[37], orientation);
        timer = createUInt(&buffer[73]);

        return true;
    }


    virtual void print(std::ostream &os) const
    {
        os << timer << ",\t" << acc(0)  << ", " << acc(1)  << ", " << acc(2)
                    << ",\t" << mag(0)  << ", " << mag(1)  << ", " << mag(2)
                    << ",\t" << gyro(0) << ", " << gyro(1) << ", " << gyro(2)
                    << ",\t" << orientation(0,0) << ", " << orientation(0,1) <<
 ", " << orientation(0,1)
                    << ",\t" << orientation(1,0) << ", " << orientation(1,1) <<
 ", " << orientation(1,2)
                    << ",\t" << orientation(2,0) << ", " << orientation(2,1) <<
 ", " << orientation(2,2);

    }

    vector acc;
    vector gyro;
    vector mag;
    matrix orientation;
    unsigned int timer;
    enum {size = 79};
};


class GX3Command
{
public:
    virtual bool sendCommand(SerialPort &serialPort) = 0;
    virtual bool checkResponse(uint8_t *buffer) = 0;
};

class SetCountinuousMode : public GX3Command
{
public:
    SetCountinuousMode(uint8_t CommandByte = 0)
    {
        mCommand[0] = SET_CONTINUOUS_MODE;
        mCommand[1] = 0xC1;
        mCommand[2] = 0x29;
        mCommand[3] = CommandByte;
    }

    bool sendCommand(SerialPort &serialPort)
    {
        serialPort.WriteRaw(mCommand, size);
        uint8_t buffer[responseSize];
        buffer[0] = serialPort.ReadByte();
        if(buffer[0] != SET_CONTINUOUS_MODE) return false;

        serialPort.ReadRaw(&buffer[1], responseSize-1);
        return checkResponse(buffer);
    }

    bool checkResponse(uint8_t *buffer)
    {

        if(buffer[0] != SET_CONTINUOUS_MODE) return false;

        if(GX3Packet::calculateChecksum(buffer, responseSize) == false)
            return false;

        if(buffer[1] != mCommand[3]) return false;

        return true;
    }

    enum {size = 4, responseSize = 8 };
    uint8_t mCommand[size];
};

class SamplingSettings : public GX3Command
{

public:
    enum FunctionSelector{
        ReturnOnly=0, Change=1,
        ChangeAndSave=2, ChangeWithoutReply=3
    };
    enum DataConditioning{
        FlagCalcOrientation      = 0x01,
        FlagEnableConingSculling = 0x02,
        FlagDefault              = 0x03,
        FlagFloatLittleEndian    = 0x10,
        FlagSuppressNaN          = 0x20,
        FlagFiniteSizeCorrection = 0x40,
        FlagDisableMag           = 0x100,
        FlagDisableMagNorthComp  = 0x400,
        FlagDisableGravComp      = 0x800,
        FlagEnableQuaternion     = 0x1000

    };

    SamplingSettings(FunctionSelector funSel, uint16_t samplingPeriod_ms = 10,
                     uint16_t dataCondFlags = SamplingSettings::FlagDefault,
                     uint8_t gyroAccFilter = 15, uint8_t magFilter = 17,
                     uint16_t upCompensation = 10, uint16_t northCompensation =
 10,
                     uint8_t magPower = 0)
    {
        mCommand[0] = SAMPLING_SETTINGS;
        mCommand[1] = 0xA8;
        mCommand[2] = 0xB9;
        mCommand[3] = (uint8_t) funSel;
        mCommand[4] = (samplingPeriod_ms >> 8);
        mCommand[5] = (samplingPeriod_ms & 0x00FF);
        mCommand[6] = (dataCondFlags >> 8);
        mCommand[7] = (dataCondFlags& 0x00FF);
        mCommand[8] = gyroAccFilter;
        mCommand[9] = magFilter;
        mCommand[10]= (upCompensation >> 8);
        mCommand[11]= (upCompensation & 0x00FF);
        mCommand[12]= (northCompensation >> 8);
        mCommand[13]= (northCompensation & 0x00FF);
```

```
            mCommand[14]= magPower;
            mCommand[15]= 0;
            mCommand[16]= 0;
            mCommand[17]= 0;
            mCommand[18]= 0;
            mCommand[19]= 0;

        }

        bool sendCommand(SerialPort &serialPort)
        {
            serialPort.WriteRaw(mCommand, size);
            uint8_t buffer[responseSize];
            buffer[0] = serialPort.ReadByte();
            if(buffer[0] != SAMPLING_SETTINGS) return false;

            serialPort.ReadRaw(&buffer[1], responseSize-1);
            return checkResponse(buffer);
        }

        bool checkResponse(uint8_t *buffer)
        {
            if(GX3Packet::calculateChecksum(buffer, responseSize) == false)
                return false;

            if(buffer[0] != SAMPLING_SETTINGS) return false;

            for(int i=1; i<11; i++)
            {
                if(buffer[i] != mCommand[i+3]) return false;
            }

            return true;
        }

        enum {size = 20, responseSize = 19};
        uint8_t mCommand[size];
    };

    }
    #endif // MESSAGES_H
```

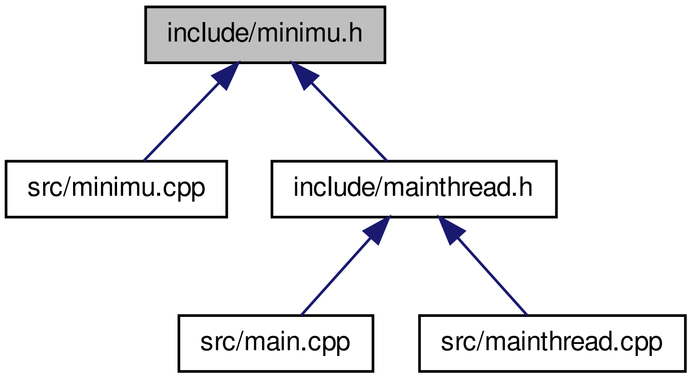
9.27 include/minimu.h File Reference

`#include <iostream>` `#include "IMU.h"` `#include "LSM303.h"` ×
`#include "L3G.h"` `#include "vector.h"` Include dependency graph for

minimu.h:



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



Classes

- class USU::MinImu

  Class to manage the communication to the Pololu MinIMU9.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

9.27.1 Detailed Description

C++ MinIMU9v2.

Author

Jan Sommer Created on: Apr 20, 2013

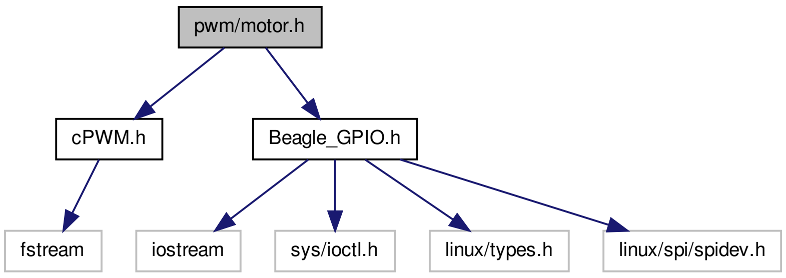
Definition in file minimu.h.

9.28 include/minimu.h

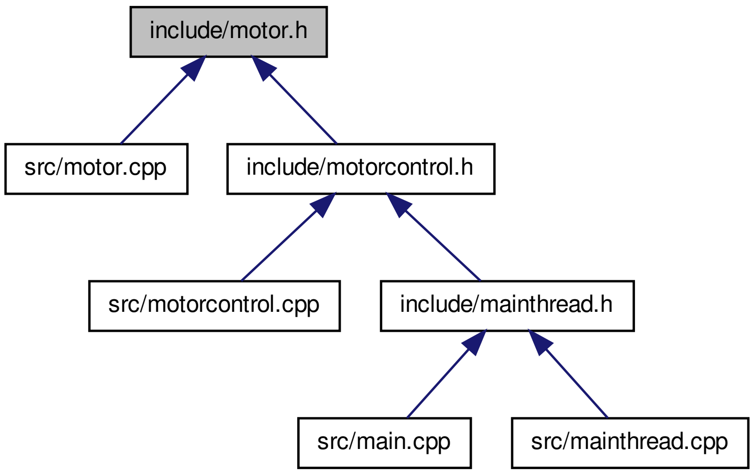
```
#ifndef MINIMU_H
#define MINIMU_H

#include <iostream>

#include "IMU.h"
#include "LSM303.h"
#include "L3G.h"
#include "vector.h"

namespace USU
{



class MinImu : public IMU
{
public:
    LSM303 compass;
    L3G gyro;
    MinImu(const char * i2cDeviceName);

    virtual vector readMag();  // In body coords, currently without scaling

    virtual vector readAcc();  // In body coords, with units = g

    virtual vector readGyro(); // In body coords, with units = degrees/sec

    virtual void enable();
};

}

#endif // MINIMU_H
```

9.29 include/motor.h File Reference

`#include "cPWM.h"` `#include "Beagle_GPIO.h"` Include dependency graph for motor.h:



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



Classes

- class USU::Motor

  Class which represents a motor.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

Typedefs

- typedef void(cPWM::∗ SetDutyCyle)(unsigned int)

  Function-pointer to the SetDutyCyle-method of cPWM class.

9.29.1 Detailed Description

Class to represent a motor

Author

Jan Sommer Created on: Apr 22, 2013

Definition in file motor.h.

9.29.2 Typedef Documentation

9.29.2.1 typedef void(cPWM::∗ **SetDutyCyle**)(unsigned int)

Function-pointer to the SetDutyCyle-method of cPWM class.

Each cPWM object has 2 channels (A and B). Each motor gets assigned to one of the channels using the corresponding function pointer.

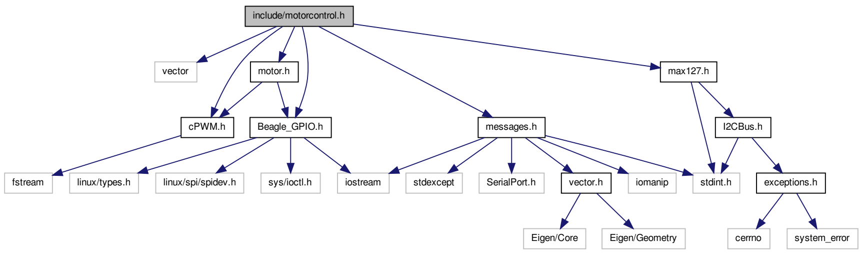
Definition at line 23 of file motor.h.

9.30 include/motor.h

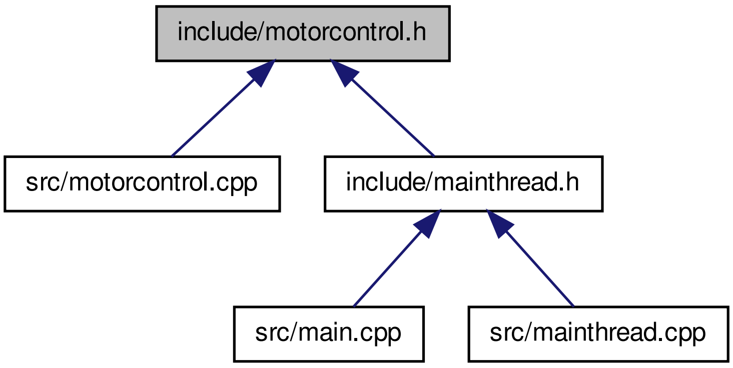
```
00001
00011 #ifndef MOTOR_H
00012 #define MOTOR_H
00013
00014 #include "cPWM.h"
00015 #include "Beagle_GPIO.h"
00016
00023 typedef void (cPWM::*SetDutyCyle)(unsigned int);
00024
00025 namespace USU
00026 {
00027
00037 class Motor
00038 {
00039 public:
00049     Motor(Beagle_GPIO& beagleGpio, Beagle_GPIO::Pins clockwise,
      Beagle_GPIO::Pins counterClockwise,
00050           cPWM& pwm, SetDutyCyle dutyCycle);
00056     void setSpeed(int speed);
00057
00063     int getSpeed() const { return mSpeed; }
00064
private:
    Beagle_GPIO& mBeagleGpio;
    Beagle_GPIO::Pins mClockwise;
    Beagle_GPIO::Pins mCounterClockwise;
    cPWM &mPwm;
    SetDutyCyle mSetDutyCycle;
    int mSpeed;
};

}

#endif // MOTOR_H
```

9.31 include/motorcontrol.h File Reference

`#include <vector>` `#include "cPWM.h"` `#include "Beagle_GPIO.h"` `#include "motor.h"` `#include "max127.h"` `#include "messages.h"` Include dependency graph for motorcontrol.h:



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



Classes

- class USU::MotorControl

    Represents the class for motor control.

Namespaces

- namespace USU

    TODO: Make some proper exceptions.

9.31.1 Detailed Description

C++ class for the calculation of the control response. Based on the PeriodicRtThread class.

Author

Jan Sommer Created on: Apr 22, 2013

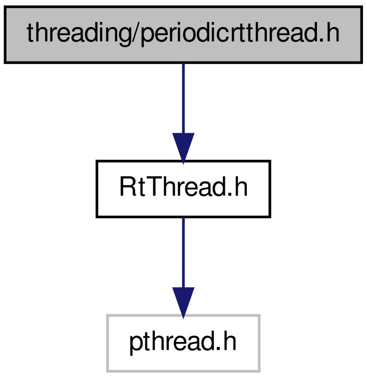
Definition in file motorcontrol.h.

9.32 include/motorcontrol.h

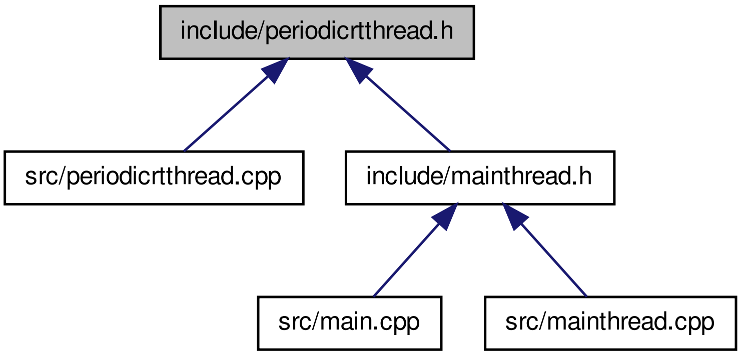
```

#ifndef MOTORCONTROL_H
#define MOTORCONTROL_H

#include <vector>

#include "cPWM.h"
#include "Beagle_GPIO.h"
#include "motor.h"
#include "max127.h"
#include "messages.h"

namespace USU
{

class MotorControl
{
public:


    MotorControl(const char* i2cDevice="/dev/i2c-3");

    virtual ~MotorControl();

    void calculateControlResponse(Quaternion state);

    void controlFromGyro(const Eigen::Vector3f &gyro);

    void setMotor(int motor, int dutyCycle);


    void getAnalog(int motor, float &aOut1, float &aOut2);

    void getAnalogs(float * aOut1, float* aOut2);

    void getDutyCycles(int* dc);

    float getPGain() const;
    void setPGain(float value);

    Eigen::Vector3f getSetValue() const;
    void setSetValue(const Eigen::Vector3f value);

private:

    cPWM mPwm1;
    cPWM mPwm2;
    Beagle_GPIO mBeagleGpio;
    Motor *mMotor[4];
    Max127 mAnalog;
    float mPGain;
    Eigen::Vector3f mSetValue;

    MotorControl(const MotorControl& thread);
    MotorControl& operator=(const MotorControl& rhs);
};

}

#endif // MOTORCONTROL_H
```

9.33 include/periodicrtthread.h File Reference

`#include "RtThread.h"` Include dependency graph for periodicrtthread.h:



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



Classes

- class USU::PeriodicRtThread

  TODO: Make some proper exceptions.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

9.33.1 Detailed Description

Small C++ wrapper class to create a realtime scheduled pthread with periodic timer events.

Author

Jan Sommer Created on: Apr 10, 2013

Definition in file periodicrtthread.h.

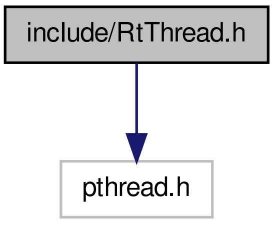
9.34 include/periodicrtthread.h

```
00001
00012 #ifndef PERIODICRTTHREAD_H
00013 #define PERIODICRTTHREAD_H
```

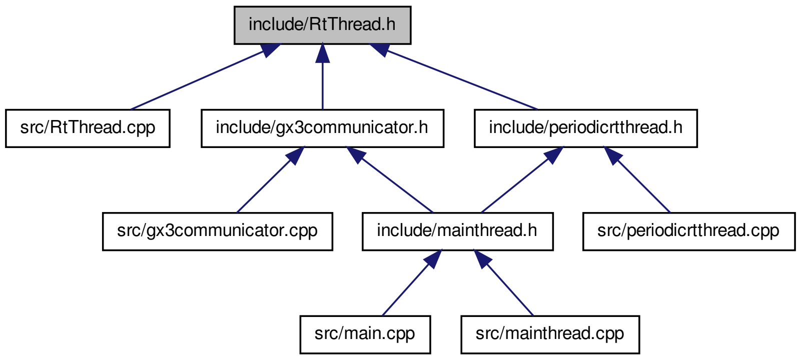
```

#include "RtThread.h"

namespace USU {


class PeriodicRtThread : public RtThread
{
private:
    int mTimerFd;
    unsigned int mMissedWakeUps;
    unsigned int mPeriod_us;
    PeriodicRtThread(const PeriodicRtThread& thread);
    PeriodicRtThread& operator=(const PeriodicRtThread& rhs);
protected:
    void makeThreadPeriodic();

    void waitPeriod();

public:
    PeriodicRtThread(int priority = 0, unsigned int period_us = 1000000);

    virtual void run() = 0;
};

}

#endif // PERIODICRTTHREAD_H
```

9.35 include/RtThread.h File Reference

`#include <pthread.h>` Include dependency graph for RtThread.h:



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



Classes

- class USU::RtThread

  Abstract wrapper class for the pthread library with RT-priority.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

9.35.1 Detailed Description

Small C++ wrapper class to create a realtime scheduled pthread

Author

Jan Sommer Created on: Apr 10, 2013

Definition in file RtThread.h.

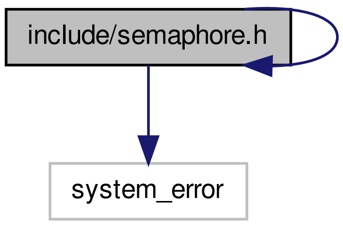
9.36 include/RtThread.h

```
#ifndef RTTHREAD_H
#define RTTHREAD_H

#include<pthread.h>

```

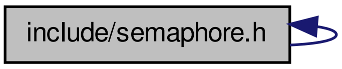
```
namespace USU {

class RtThread
{

private:
    pthread_attr_t mAttr;
    int mPriority;
    RtThread(const RtThread& thread);
    RtThread& operator=(const RtThread& rhs);
protected:
    pthread_t mId;
    bool mStarted;
    void *mArgs;
    static void *exec(void * thr);

public:
    RtThread(int priority = 0);

    virtual ~RtThread();

    pthread_t getThreadId() const;

    int getPriority() const;

    void start(void * args = NULL);

    bool join(int timeout_ms = 0);

    virtual void run() = 0;
};

}

#endif // RTTHREAD_H
```

9.37 include/semaphore.h File Reference

`#include <semaphore.h>` `#include <system_error>` Include dependency graph for semaphore.h:



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



Classes

- class USU::Semaphore

  Wrapper class for semaphores.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

9.37.1 Detailed Description

Small wrapper class for semaphore

Author

Jan Sommer Created on: Apr 30, 2013

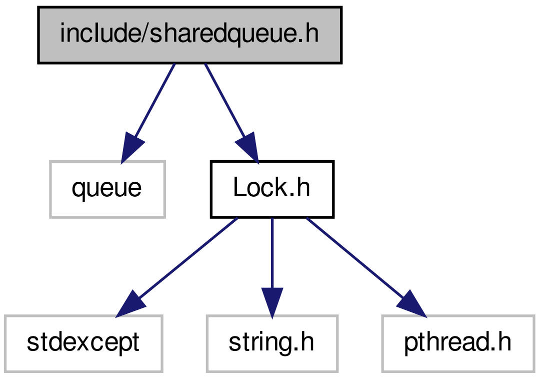
Definition in file semaphore.h.

9.38 include/semaphore.h

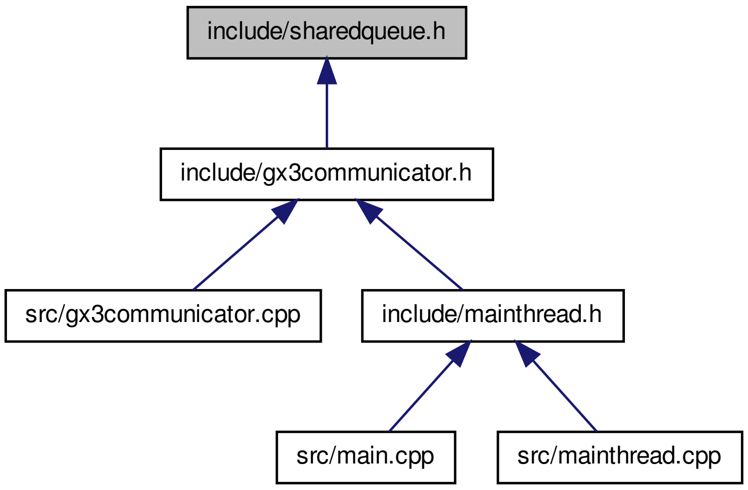
```
#ifndef SEMAPHORE_H
#define SEMAPHORE_H

#include <semaphore.h>
#include <system_error>

namespace USU
{


class Semaphore
{
private:

    sem_t mSem;
    Semaphore(const Semaphore& arg);
    Semaphore& operator=(const Semaphore& rhs);
public:

    Semaphore();
    virtual ~Semaphore();
    void post();
    void wait();

    bool tryWait();
};

Semaphore::Semaphore()
{
    int ret = sem_init(&mSem, 0, 0);
    if(ret != 0)
    {
        throw std::system_error(ret, std::system_category());
    }
}

Semaphore::~Semaphore()
{
    int ret = sem_close(&mSem);
    if(ret != 0)
    {
        throw std::system_error(ret, std::system_category());
    }
}

inline
void Semaphore::post()
{
    sem_post(&mSem);
}

inline
void Semaphore::wait()
{
    sem_wait(&mSem);
}

inline
bool Semaphore::tryWait()
{
    return (!sem_trywait(&mSem) ? true : false);
}

}

#endif // SEMAPHORE_H
```

9.39 include/sharedqueue.h File Reference

`#include <queue>` `#include "Lock.h"` Include dependency graph for sharedqueue.h:



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



Classes

- class USU::SharedQueue< T >

  Wrapper class to make std::queue thread safe.

Namespaces

- namespace USU

  TODO: Make some proper exceptions.

9.39.1 Detailed Description

Small wrapper class to make std::queue thread safe in the sense of the single producer, single consumer problem.

Author

Jan Sommer Created on: May 2, 2013

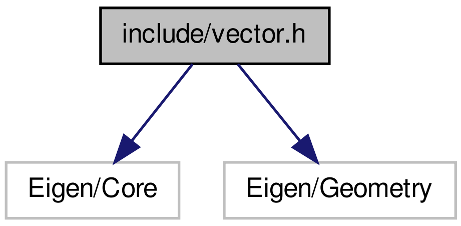
Definition in file sharedqueue.h.

9.40 include/sharedqueue.h

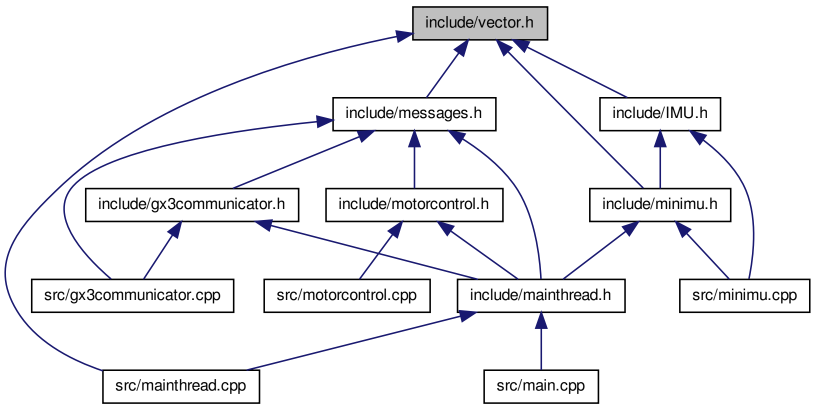
```

#ifndef SHAREDQUEUE_H
#define SHAREDQUEUE_H

#include <queue>
using std::queue;

#include "Lock.h"

namespace USU
{
template <class T>
class SharedQueue
{
public:
//    SharedQueue();

    void push(const T& newElement)
    {
        ScopedLock scLock(mLock);
        mQueue.push(newElement);
    }

    void pop()
    {
        ScopedLock scLock(mLock);
        mQueue.pop();
    }

    T& front()
    {
        ScopedLock scLock(mLock);
        return mQueue.front();
    }

    bool isEmpty()
    {
        ScopedLock scLock(mLock);
        return mQueue.empty();
    }

    int size()
    {
        ScopedLock scLock(mLock);
        return mQueue.size();
    }

private:
    Lock mLock;
    queue<T> mQueue;
};

}

#endif // SHAREDQUEUE_H
```

9.41 include/vector.h File Reference

`#include "Eigen/Core"` `#include "Eigen/Geometry"` Include dependency graph for vector.h:



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



Typedefs

- typedef Eigen::Vector3f vector
- typedef Eigen::Vector3i int_vector
- typedef Eigen::Matrix3f matrix
- typedef Eigen::Quaternionf quaternion

9.41.1 Typedef Documentation

9.41.1.1 typedef Eigen::Vector3i int_vector

Definition at line 7 of file vector.h.

9.41.1.2 typedef Eigen::Matrix3f matrix

Definition at line 8 of file vector.h.

9.41.1.3 typedef Eigen::Quaternionf quaternion

Definition at line 9 of file vector.h.

9.41.1.4 typedef Eigen::Vector3f vector

Definition at line 6 of file vector.h.

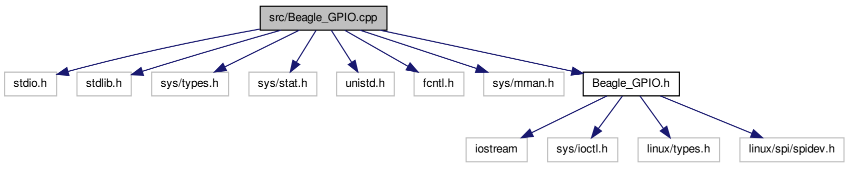
9.42 include/vector.h

```
#ifndef _vector_h
```

```
#define _vector_h

#include "Eigen/Core"
#include "Eigen/Geometry"
typedef Eigen::Vector3f vector;
typedef Eigen::Vector3i int_vector;
typedef Eigen::Matrix3f matrix;
typedef Eigen::Quaternionf quaternion;

static inline vector vector_from_ints(int (*ints)[3])
{
    return vector((float)(*ints)[0], (float)(*ints)[1], (float)(*ints)[2]);
}

static inline int_vector int_vector_from_ints(int (*ints)[3])
{
    return int_vector((*ints)[0], (*ints)[1], (*ints)[2]);
}

#endif
```

9.43 src/Beagle_GPIO.cpp File Reference

`#include <stdio.h>` `#include <stdlib.h>` `#include <sys/types.h>` `#include <sys/stat.h>` `#include <unistd.h>` `#include <fcntl.h>` `#include <sys/mman.h>` `#include "Beagle_GPIO.h"`
Include dependency graph for Beagle_GPIO.cpp:



9.44 src/Beagle_GPIO.cpp

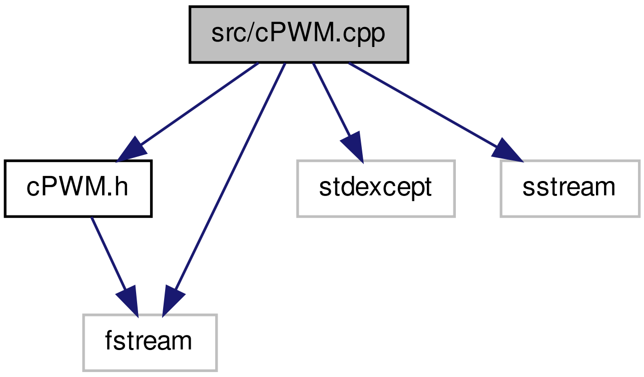
```
/******************************
 ** Beagle Bone GPIO Library **
 **                          **
 **      Francois Sugny      **
 **         01/07/12         **
 **                          **
 **          v1.0            **
 ******************************/

//=======================================================
//=======================================================

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/mman.h>

//=======================================================
//=======================================================

#include "Beagle_GPIO.h"

//=======================================================
//=======================================================

const int Beagle_GPIO::GPIO_Pin_Bank[] =
{
               -1, -1,  1,  1,  1,           // P8_1  -> P8_5
                1,  2,  2,  2,  2,           // P8_6  -> P8_10
                1,  1,  0,  0,  1,           // P8_11 -> P8_15
                1,  0,  2,  0,  1,           // P8_16 -> P8_20
                1,  1,  1,  1,  1,           // P8_21 -> P8_25
                1,  2,  2,  2,  2,           // P8_26 -> P8_30
                0,  0,  0,  2,  0,           // P8_31 -> P9_35
                2,  2,  2,  2,  2,           // P8_36 -> P8_40
                2,  2,  2,  2,  2,           // P8_41 -> P8_45
                2,                                          // P8_46
               -1, -1, -1, -1, -1,           // P9_1  -> P9_5
               -1, -1, -1, -1, -1,           // P9_6  -> P9_10
                0,  1,  0,  1,  1,           // P9_11 -> P9_15
                1,  0,  0,  0,  0,           // P9_16 -> P9_20
                0,  0,  1,  0,  3,           // P9_21 -> P9_25
                0,  3,  3,  3,  3,           // P9_26 -> P9_30
                3, -1, -1, -1, -1,           // P9_31 -> P9_35
               -1, -1, -1, -1, -1,           // P9_36 -> P9_40
                0,  0, -1, -1, -1,           // P9_41 -> P9_45
               -1                                           // P9_46
};

//=======================================================
//=======================================================

const int Beagle_GPIO::GPIO_Pin_Id[] =
{
               -1, -1,  6,  7,  2,           // P8_1  -> P8_5
                3,  2,  3,  5,  4,           // P8_6  -> P8_10
               13, 12, 23, 26, 15,           // P8_11 -> P8_15
               14, 27,  1, 22, 31,           // P8_16 -> P8_20
               30,  5,  4,  1,  0,           // P8_21 -> P8_25
               29, 22, 24, 23, 25,           // P8_26 -> P8_30
               10, 11,  9, 17,  8,           // P8_31 -> P9_35
               16, 14, 15, 12, 13,           // P8_36 -> P8_40
               10, 11,  8,  9,  6,           // P8_41 -> P8_45
                7,                                          // P8_46
               -1, -1, -1, -1, -1,           // P9_1  -> P9_5
               -1, -1, -1, -1, -1,           // P9_6  -> P9_10
               30, 28, 31, 18, 16,           // P9_11 -> P9_15
               19,  5,  4, 13, 12,           // P9_16 -> P9_20
                3,  2, 17, 15, 21,           // P9_21 -> P9_25
               14, 19, 17, 15, 16,           // P9_26 -> P9_30
               14, -1, -1, -1, -1,           // P9_31 -> P9_35
               -1, -1, -1, -1, -1,           // P9_36 -> P9_40
               20,  7, -1, -1, -1,           // P9_41 -> P9_45
               -1                                           // P9_46
};

//=======================================================
//=======================================================

// Pad Control Register
const unsigned long Beagle_GPIO::GPIO_Pad_Control[] =
{
               0x0000, 0x0000, 0x0818, 0x081C, 0x0808,      // P8_1  -> P8_5
               0x080C, 0x0890, 0x0894, 0x089C, 0x0898,      // P8_6  -> P8_10
               0x0834, 0x0830, 0x0824, 0x0828, 0x083C,      // P8_11 -> P8_15
               0x0838, 0x082C, 0x088C, 0x0820, 0x0884,      // P8_16 -> P8_20
               0x0880, 0x0814, 0x0810, 0x0804, 0x0800,      // P8_21 -> P8_25
               0x087C, 0x08E0, 0x08E8, 0x08E4, 0x08EC,      // P8_26 -> P8_30
               0x08D8, 0x08DC, 0x08D4, 0x08CC, 0x08D0,      // P8_31 -> P8_35
               0x08C8, 0x08C0, 0x08C4, 0x08B8, 0x08BC,      // P8_36 -> P8_40
               0x08B0, 0x08B4, 0x08A8, 0x08AC, 0x08A0,      // P8_41 -> P8_45
               0x08A4,
// P8_46
               0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // P9_1  -> P9_5
               0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // P9_6  -> P9_10
               0x0870, 0x0878, 0x0874, 0x0848, 0x0840,      // P9_11 -> P9_15
               0x084C, 0x095C, 0x0958, 0x097C, 0x0978,      // P9_16 -> P9_20
               0x0954, 0x0950, 0x0844, 0x0984, 0x09AC,      // P9_21 -> P9_25
               0x0980, 0x09A4, 0x099C, 0x0994, 0x0998,      // P9_26 -> P9_30
               0x0990, 0x0000, 0x0000, 0x0000, 0x0000,      // P9_31 -> P9_35
               0x0000, 0x0000, 0x0000, 0x0000, 0x0000,      // P9_36 -> P9_40
               0x09B4, 0x0964, 0x0000, 0x0000, 0x0000,      // P9_41 -> P9_45
               0x0000
// P9_46
};

//=======================================================
//=======================================================

const unsigned long Beagle_GPIO::GPIO_Control_Module_Registers = 0x44E10000;

//=======================================================
//=======================================================

const unsigned long Beagle_GPIO::GPIO_Base[] =
{
               0x44E07000,    // GPIO0
               0x4804C000,    // GPIO1
               0x481AC000,    // GPIO2
               0x481AE000     // GPIO3
};

//=======================================================
//=======================================================

Beagle_GPIO::Beagle_GPIO()
{
               GPIO_PRINT( "Beagle_GPIO::Beagle_GPIO()" );

               // Not initialized by default
               m_active = false;

               // Opening /dev/mem first
               GPIO_PRINT( "Opening /dev/mem" );
               m_gpio_fd = open( "/dev/mem", O_RDWR | O_SYNC );
               if ( m_gpio_fd < 0 )
               {
                              GPIO_ERROR( "Cannot open /dev/mem" );
                              return;
               }

               // Map Control Module
               m_controlModule = (unsigned long *)mmap( NULL, 0x1FFF, PROT_READ
 | PROT_WRITE, MAP_SHARED, m_gpio_fd, GPIO_Control_Module_Registers );
               if ( m_controlModule == MAP_FAILED )
               {
                              GPIO_ERROR( "Control Module Mapping failed" );
                              return;
               }

               // Now mapping the GPIO registers
               for ( int i=0; i<4; ++i)
               {
                              // Map a GPIO bank
                              m_gpio[i] = (unsigned long *)mmap( NULL, 0xFFF,
PROT_READ | PROT_WRITE, MAP_SHARED, m_gpio_fd, GPIO_Base[i] );
                              if ( m_gpio[i] == MAP_FAILED )
                              {
                                             GPIO_ERROR( "GPIO Mapping failed
 for GPIO Module " << i );
                                             return;
                              }
               }

               // Init complete and successfull
               m_active = true;
```

```

                GPIO_PRINT( "Beagle GPIO Initialized" );
}

//=======================================================
//=======================================================

Beagle_GPIO::~Beagle_GPIO()
{
                //GPIO_PRINT( "BeAGLe_GPIO::~Beagle_GPIO()" );
                if ( m_active && m_gpio_fd)
                                close( m_gpio_fd );
}

//=======================================================
//=======================================================

// Configure pin as input/output
Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::configurePin( unsigned short _pin, Beagle_GPIO_Direction _direction )
{
                if ( !m_active )
                                return kFail;

    gp_assert(GPIO_Pin_Bank[_pin]>=0);
    gp_assert(GPIO_Pin_Id[_pin]>=0);

                // Set Pin as GPIO on the pad control
                m_controlModule[GPIO_Pad_Control[_pin]/4] |= 0x07;

                unsigned long v = 0x1 << GPIO_Pin_Id[_pin];

                if ( _direction == kINPUT)
                {
                                m_gpio[GPIO_Pin_Bank[_pin]][kOE/4] |= v;
                }
                else
                {
                                m_gpio[GPIO_Pin_Bank[_pin]][kOE/4] &= ~v;
                }

                // Disable Interrupts by default
                m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_0/4] |= v;
                m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_1/4] |= v;

                return kSuccess;
}

//=======================================================
//=======================================================

// Enable/Disable interrupts for the pin
Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::enablePinInterrupts( unsigned short _pin, bool _enable )
{
                if ( !m_active )
                                return kFail;

    gp_assert(GPIO_Pin_Bank[_pin]>=0);
    gp_assert(GPIO_Pin_Id[_pin]>=0);

                // Set Pin as GPIO on the pad control
                m_controlModule[GPIO_Pad_Control[_pin]/4] |= 0x07;

                unsigned long v = 0x1 << GPIO_Pin_Id[_pin];

                if ( _enable )
                {
                                m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_SET_0/4] |= v;
                                m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_SET_1/4] |= v;
                }
                else
                {
                                m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_0/4] |
= v;
                                m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_1/4] |
= v;
                }

                return kSuccess;

}

//=======================================================
//=======================================================

// Write a value to a pin
Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::writePin( unsigned short _pin,
unsigned char _value )
{
    gp_assert(GPIO_Pin_Bank[_pin]>=0);
    gp_assert(GPIO_Pin_Id[_pin]>=0);

                unsigned long v = (_value & 0x01) << GPIO_Pin_Id[_pin];
                unsigned long mask = 0x1 << GPIO_Pin_Id[_pin];

                // Remove bit
                m_gpio[GPIO_Pin_Bank[_pin]][kDATAOUT/4] &= ~mask;
                // Assign new bit value
                m_gpio[GPIO_Pin_Bank[_pin]][kDATAOUT/4] |= v;

                return kSuccess;
}

//=======================================================
//=======================================================

// Read a value from a pin
unsigned char Beagle_GPIO::readPin( unsigned short _pin )
{
    gp_assert(GPIO_Pin_Bank[_pin]>=0);
    gp_assert(GPIO_Pin_Id[_pin]>=0);

                unsigned long bit = GPIO_Pin_Id[_pin];
                return (m_gpio[GPIO_Pin_Bank[_pin]][kDATAIN/4] & (0x1 << bit)) >
> bit;
}

//=======================================================
//=======================================================

// Default SPI Device for the beaglebone
static const char *spi_device = "/dev/spidev2.0";

// Open SPI Channel
void Beagle_GPIO::openSPI( unsigned char _mode,
                                        unsigned char _bits,
                                        unsigned long _speed,
                                        unsigned short _delay )
{
                GPIO_PRINT( "Opening SPI Device" );
                m_spi_fd = open( spi_device, O_RDWR );
                if ( m_spi_fd < 0 )
                {
                                GPIO_ERROR( "Error opening SPI Device" );
                                return;
                }

                int ret = 0;

                // Save settings
                m_spi_mode = _mode;
                m_spi_bits = _bits;
                m_spi_speed = _speed;
                m_spi_delay = _delay;

                m_spi_buffer_rx = new unsigned char[65536];

                // SPI Mode
                ret = ioctl(m_spi_fd, SPI_IOC_WR_MODE, &m_spi_mode);
                if (ret == -1)
                {
                                GPIO_ERROR( "Error setting SPI Mode");
                                return;
                }

                ret = ioctl(m_spi_fd, SPI_IOC_RD_MODE, &m_spi_mode);
                if (ret == -1)
                {
                                GPIO_ERROR( "Error getting SPI Mode");
                                return;
                }

                // SPI Bits Per Word
                ret = ioctl(m_spi_fd, SPI_IOC_WR_BITS_PER_WORD, &m_spi_bits);
                if (ret == -1)
                {
                                GPIO_ERROR( "Error setting SPI Bits Per Word");
                                return;
                }

                ret = ioctl(m_spi_fd, SPI_IOC_RD_BITS_PER_WORD, &m_spi_bits);
                if (ret == -1)
                {
                                GPIO_ERROR( "Error getting SPI Bits Per Word");
                                return;
                }

                // SPI Max Speed
                ret = ioctl(m_spi_fd, SPI_IOC_WR_MAX_SPEED_HZ, &m_spi_speed);
                if (ret == -1)
                {
                                GPIO_ERROR( "Error setting SPI Max Speed");
                                return;
                }

                ret = ioctl(m_spi_fd, SPI_IOC_RD_MAX_SPEED_HZ, &m_spi_speed);
                if (ret == -1)
                {
                                GPIO_ERROR( "Error getting SPI Max Speed");
                                return;
                }

                GPIO_PRINT( "SPI Mode : " << std::hex << (int)(m_spi_mode) );
                GPIO_PRINT( "SPI Bits Per Word : " << std::dec << (int)(m_spi_bits) );
                GPIO_PRINT( "SPI Max Speed : " << std::dec << m_spi_speed );
                GPIO_PRINT( "SPI Delay : " << std::dec << m_spi_delay );
                GPIO_PRINT( "SPI Opened" );
}

//=======================================================
//=======================================================

// Close SPI Channel
void Beagle_GPIO::closeSPI()
{
                if ( m_spi_fd >= 0)
                {
                                GPIO_PRINT( "Closing SPI Device" );
                                close( m_spi_fd );
                                delete [] m_spi_buffer_rx;
                }
}

//=======================================================
//=======================================================

// Send SPI Buffer
void Beagle_GPIO::sendSPIBuffer( unsigned long _buffer, int _size )
```

```
{
    gp_assert( m_spi_fd >= 0 );
    gp_assert( _buffer > 0 );
    gp_assert( _size > 0 );

                m_spi_ioc_tr.tx_buf = _buffer;
                m_spi_ioc_tr.rx_buf = (unsigned long)(m_spi_buffer_rx);
                m_spi_ioc_tr.len = _size;
                m_spi_ioc_tr.delay_usecs = m_spi_delay;
                m_spi_ioc_tr.speed_hz = m_spi_speed;
                m_spi_ioc_tr.bits_per_word = m_spi_bits;

                if ( ioctl( m_spi_fd, SPI_IOC_MESSAGE(1), &m_spi_ioc_tr ) < 1 )
                {
                               GPIO_ERROR( "Cannot send SPI Buffer, size=" <<
std::dec << _size );
                               return;
                }
}

//=======================================================
//=======================================================


```

9.45 src/cPWM.cpp File Reference

`#include "cPWM.h"` `#include <stdexcept>` `#include <fstream>` × `#include <sstream>` Include dependency graph for cPWM.cpp:



9.45.1 Detailed Description

Simple C++ class wrapper for beaglebone PWM eHRPWM interface

Author

claus Created on: Jun 13, 2012 Author: claus http://quadrotordiaries.-blogspot.com

Definition in file cPWM.cpp.

9.46 src/cPWM.cpp

```
// $Id$
// $Log$


#include "cPWM.h"
#include <stdexcept>
#include <fstream>
#include <sstream>

cPWM::cPWM(int id)
{
               cPWM::id = id;

    std::stringstream sysfsfile_dutyA_ns;
    std::stringstream sysfsfile_dutyA_percent;

    std::stringstream sysfsfile_dutyB_ns;
    std::stringstream sysfsfile_dutyB_percent;

    std::stringstream sysfsfile_period_ns;
    std::stringstream sysfsfile_period_freq;

               std::stringstream sysfsfile_polarityA;
               std::stringstream sysfsfile_runA;
    std::stringstream sysfsfile_requestA;

    std::stringstream sysfsfile_polarityB;
               std::stringstream sysfsfile_runB;
               std::stringstream sysfsfile_requestB;

    sysfsfile_dutyA_ns << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_DUTY_NS;
    sysfsfile_dutyA_percent << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_DUTY_PERCENT;

    sysfsfile_dutyB_ns << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_DUTY_NS;
    sysfsfile_dutyB_percent << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_DUTY_PERCENT;

    sysfsfile_period_ns << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_PERIOD_NS;
    sysfsfile_period_freq << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_PERIOD_FREQ;

               sysfsfile_polarityA << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_POLARITY;
               sysfsfile_runA << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_RUN;
               sysfsfile_requestA << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_REQUEST;

    sysfsfile_polarityB << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_POLARITY;
               sysfsfile_runB << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_RUN;
               sysfsfile_requestB << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_REQUEST;

    sysfsfid_dutyA_ns.open(sysfsfile_dutyA_ns.str().c_str());
    sysfsfid_dutyA_percent.open(sysfsfile_dutyA_percent.str().c_str());

    sysfsfid_dutyB_ns.open(sysfsfile_dutyB_ns.str().c_str());
    sysfsfid_dutyB_percent.open(sysfsfile_dutyB_percent.str().c_str());

    sysfsfid_period_ns.open(sysfsfile_period_ns.str().c_str());
    sysfsfid_period_freq.open(sysfsfile_period_freq.str().c_str());

                sysfsfid_polarityA.open(sysfsfile_polarityA.str().c_str());
                sysfsfid_runA.open(sysfsfile_runA.str().c_str());

    sysfsfid_requestA.open(sysfsfile_requestA.str().c_str());
                sysfsfid_polarityB.open(sysfsfile_polarityB.str().c_str());

    sysfsfid_runB.open(sysfsfile_runB.str().c_str());
                sysfsfid_requestB.open(sysfsfile_requestB.str().c_str());
}

void cPWM::DutyA_ns(unsigned int nanoseconds)
{
    if(nanoseconds > cPWM::period)
        throw std::out_of_range("DutyA_ns: ");

    cPWM::dutyA = nanoseconds;
    sysfsfid_dutyA_ns << nanoseconds << std::endl;
}

void cPWM::DutyA_percent(unsigned int percent)
{
    if(percent > 100)
        throw std::out_of_range("DutyA_percent: ");

        sysfsfid_dutyA_percent << percent << std::endl;
}

void cPWM::DutyB_ns(unsigned int nanoseconds)
{
    if(nanoseconds > cPWM::period)
        throw std::out_of_range("DutyB_ns: ");

    cPWM::dutyB = nanoseconds;
    sysfsfid_dutyB_ns << nanoseconds << std::endl;
}


void cPWM::DutyB_percent(unsigned int percent)
{
    if(percent > 100)
        throw std::out_of_range("DutyB_percent: ");

    sysfsfid_dutyB_percent << percent << std::endl;
}


void cPWM::Period_ns(unsigned int nanoseconds)
{
        cPWM::period  = nanoseconds;
        cPWM::freq_Hz = 1000000000 / nanoseconds;
        sysfsfid_period_ns << nanoseconds << std::endl;
}

void cPWM::Period_freq(unsigned int freq_Hz)
{
        cPWM::freq_Hz = freq_Hz;
        cPWM::period  = 1000000000 / freq_Hz;
        sysfsfid_period_freq << freq_Hz<< std::endl;
}

void cPWM::PolarityA(Polarity polarity)
{
        switch (polarity)
        {
        case ActiveHigh:  sysfsfid_polarityA << 1 << std::endl;
                          break;
        case ActiveLow:   sysfsfid_polarityA << 0 << std::endl;
```

```
                          break;
        }
        cPWM::polarityA = polarity;
}

void cPWM::RunA()
{
               sysfsfid_runA << "1" << std::endl;
               cPWM::runA = 1;
}

void cPWM::StopA()
{
               sysfsfid_runA << "0" << std::endl;
               cPWM::runA = 0;
}

void cPWM::PolarityB(Polarity polarity)
{
    switch (polarity)
    {
    case ActiveHigh:  sysfsfid_polarityB << 1 << std::endl;
                      break;
    case ActiveLow:   sysfsfid_polarityB << 0 << std::endl;
                      break;
    }
    cPWM::polarityB = polarity;
}

void cPWM::RunB()
{
               cPWM::runB = 1;
               sysfsfid_runB << "1" << std::endl;
}

void cPWM::StopB()
{
               cPWM::runB = 0;
               sysfsfid_runB << "0" << std::endl;
}

cPWM::~cPWM()
{
               sysfsfid_runA << "0" << std::endl;

               sysfsfid_runB << "0" << std::endl;
}
```

9.47 src/gx3communicator.cpp File Reference

`#include <stdint.h>` `#include <iostream>` `#include <iomanip>` × `#include <stdexcept>` `#include <sys/time.h>` `#include "gx3communicator.h"` `#include "messages.h"` Include dependency graph for gx3communicator.cpp:

9.47.1 Detailed Description

Contains the thread which handles the communication to the 3DM-GX3-25.

Author

Jan Sommer Created on: Apr 26, 2013

Definition in file gx3communicator.cpp.

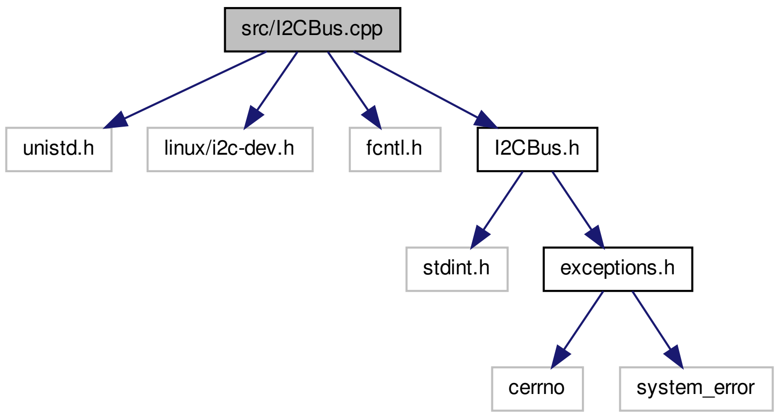
9.48 src/gx3communicator.cpp

```

#include <stdint.h>
#include <iostream>
#include <iomanip>
#include <stdexcept>

#include <sys/time.h>

#include "gx3communicator.h"
using namespace USU;

#include "messages.h"

//int timeval_subtract (struct timeval * result, struct timeval * x, struct
 timeval * y)
//{
//    /* Perform the carry for the later subtraction by updating y. */
//    if (x->tv_usec < y->tv_usec) {
//        int nsec = (y->tv_usec - x->tv_usec) / 1000000 + 1;
//        y->tv_usec -= 1000000 * nsec;
//        y->tv_sec += nsec;
//    }
//    if (x->tv_usec - y->tv_usec > 1000000) {
//        int nsec = (x->tv_usec - y->tv_usec) / 1000000;
//        y->tv_usec += 1000000 * nsec;
//        y->tv_sec -= nsec;
//    }

//    /* Compute the time remaining to wait.
//          tv_usec is certainly positive. */
//    result->tv_sec = x->tv_sec - y->tv_sec;
//    result->tv_usec = x->tv_usec - y->tv_usec;

//    /* Return 1 if result is negative. */
//    return x->tv_sec < y->tv_sec;
//}

GX3Communicator::GX3Communicator(int priority, const char *serialDevice,
SerialPort::BaudRate baudRate)
    :RtThread(priority), mSerialPort(serialDevice), mBaudRate(baudRate),
mKeepRunning(false)
{

}

void GX3Communicator::initialize()
{
    mSerialPort.Open(mBaudRate);
    if(mSerialPort.IsOpen() == false)
        throw std::runtime_error("Opening SerialPort failed");

    /*
       Set up the 3DM-GX25 with the following settings (different from
 default):
        - Data rate 50 Hz
        - Enable little endian for floating points
     */
    SamplingSettings initSettings(SamplingSettings::Change,  20,
                                  SamplingSettings::FlagDefault |
SamplingSettings::FlagFloatLittleEndian
                                    | SamplingSettings::FlagEnableQuaternion);

```

```cpp
    if(initSettings.sendCommand(mSerialPort) == false)
        throw std::runtime_error("Setting SamplingSettings failed");

}

void GX3Communicator::run()
{

    mKeepRunning = true;

    // Activate Continuous mode
    SetCountinuousMode setCont(ACC_ANG_MAG_VEC);
    if(setCont.sendCommand(mSerialPort) == false)
        std::cerr << " Set continuous mode failed " << std::endl;

//    struct timeval start, now, elapsed;

//    gettimeofday(&start, NULL);
    while(mKeepRunning)
    {
        packet_ptr data(new AccAngMag);
        if(data->readFromSerial(mSerialPort))
        {
            mQueue.push(data);
            //        gettimeofday(&now, NULL);
            //        timeval_subtract(&elapsed, &now, &start);
//                    unsigned long long timestamp = elapsed.tv_sec * 1000 + elapsed.tv_usec / 1000; // in ms since start
//                    std::cout << (*data) << std::endl;
        }
        else
        {
//            std::cout << "readFromSerial failed" << std::endl;
            //            throw std::runtime_error("Getting PackageData failed"); /// TODO: Error?

        }
    }

    std::cerr << "GX3COMMUNICATOR: Got signal to terminate" << std::endl;
    std::cerr << "GX3COMMUNICATOR: Stopping IMU continuous mode..." << std::endl;
    // Stop continuous mode
    setCont.mCommand[3] = 0;
    if(setCont.sendCommand(mSerialPort) == false)
        ;

    std::cerr << "GX3COMMUNICATOR: IMU continuous mode stopped" << std::endl;
    std::cerr << "GX3COMMUNICATOR: Terminating now..." << std::endl;
}


```

9.49 src/I2CBus.cpp File Reference

`#include <unistd.h>` `#include <linux/i2c-dev.h>` `#include <fcntl.h>` `#include "I2CBus.h"` Include dependency graph for I2CBus.cpp:

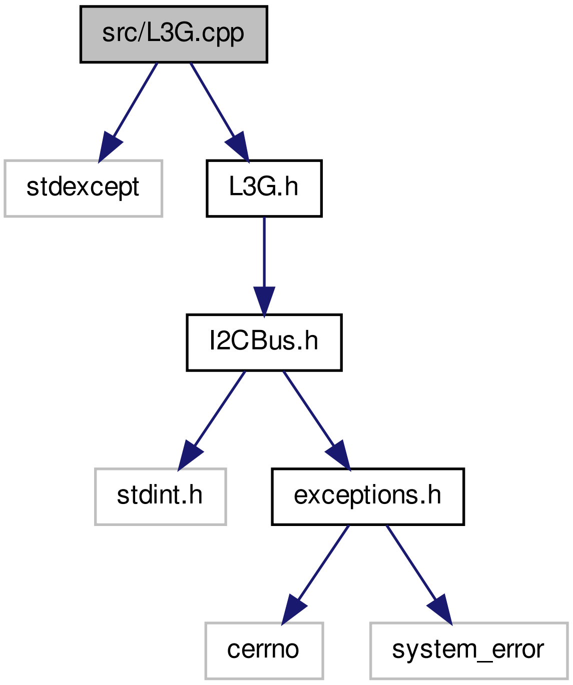


9.50 src/I2CBus.cpp

```
//#include <sys/ioctl.h>
#include <unistd.h>
#include <linux/i2c-dev.h>
#include <fcntl.h>

#include "I2CBus.h"

I2CBus::I2CBus(const char * deviceName)
{
    fd = open(deviceName, O_RDWR);
    if (fd == -1)
    {
        throw posix_error("Failed to open I2C device.");
    }
}

I2CBus::~I2CBus()
{
    close(fd);
}

void I2CBus::addressSet(uint8_t address)
{
    int result = ioctl(fd, I2C_SLAVE, address);
    if (result == -1)
    {
        throw posix_error("Failed to set address.");
    }
}

void I2CBus::writeByte(uint8_t command, uint8_t data)
{
    int result = i2c_smbus_write_byte_data(fd, command, data);
    if (result == -1)
    {
        throw posix_error("Failed to write byte to I2C.");
    }
}

void I2CBus::writeByte(uint8_t data)
{
    int result = i2c_smbus_write_byte(fd, data);
    if (result == -1)
    {
        throw posix_error("Failed to write raw byte to I2C.");
    }
}

uint8_t I2CBus::readByte(uint8_t command)
{
    int result = 0;
    result = i2c_smbus_read_byte_data(fd, command);
    if (result == -1)
    {
//        throw posix_error("Failed to read byte from I2C.");
        return 0;
    }
    return (uint8_t) result;
}

uint8_t I2CBus::readByte()
{
    int result = i2c_smbus_read_byte(fd);
    if (result == -1)
    {
        throw posix_error("Failed to read raw byte from I2C.");
    }
    return result;
}

uint16_t I2CBus::readWord(uint8_t command)
{
    int result = i2c_smbus_read_word_data(fd, command);
    if (result == -1)
    {
        throw posix_error("Failed to read word from I2C.");
    }
    return result;
}

uint16_t I2CBus::readWord()
{
    uint16_t temp;
    int result = read(fd, &temp, sizeof(uint16_t) );
    if(result == -1)
    {
        throw posix_error("Failed to read raw word from I2C");
    }
    return temp;
}

int I2CBus::tryReadByte(uint8_t command)
{
    return i2c_smbus_read_byte_data(fd, command);
}


void I2CBus::readBlock(uint8_t command, uint8_t size, uint8_t * data)
{
    int result = i2c_smbus_read_i2c_block_data(fd, command, size, data);
    if (result != size)
    {
        throw posix_error("Failed to read block from I2C.");
    }
}
```

9.51 src/L3G.cpp File Reference

`#include <stdexcept>` `#include "L3G.h"` Include dependency graph for L3G.cpp:



Defines

- #define L3G4200D_ADDRESS_SA0_LOW (0xD0 >> 1)
- #define L3G4200D_ADDRESS_SA0_HIGH (0xD2 >> 1)
- #define L3GD20_ADDRESS_SA0_LOW (0xD4 >> 1)
- #define L3GD20_ADDRESS_SA0_HIGH (0xD6 >> 1)

9.51.1 Define Documentation

9.51.1.1 #define L3G4200D_ADDRESS_SA0_HIGH (0xD2 >> 1)

Definition at line 5 of file L3G.cpp.

9.51.1.2 #define L3G4200D_ADDRESS_SA0_LOW (0xD0 >> 1)

Definition at line 4 of file L3G.cpp.

9.51.1.3 #define L3GD20_ADDRESS_SA0_HIGH (0xD6 >> 1)

Definition at line 7 of file L3G.cpp.

9.51.1.4 #define L3GD20_ADDRESS_SA0_LOW (0xD4 >> 1)

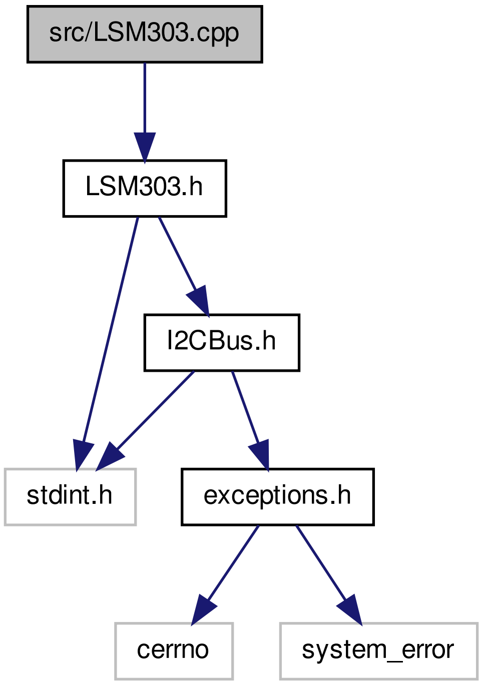
Definition at line 6 of file L3G.cpp.

9.52 src/L3G.cpp

```
00001 #include <stdexcept>
00002 #include "L3G.h"
00003
00004 #define L3G4200D_ADDRESS_SA0_LOW  (0xD0 >> 1)
00005 #define L3G4200D_ADDRESS_SA0_HIGH (0xD2 >> 1)
00006 #define L3GD20_ADDRESS_SA0_LOW    (0xD4 >> 1)
00007 #define L3GD20_ADDRESS_SA0_HIGH   (0xD6 >> 1)
00008
00009 L3G::L3G(const char * i2cDeviceName) : i2c(i2cDeviceName)
00010 {
00011 }
00012
00013 void L3G::detectAddress()
00014 {
00015     i2c.addressSet(L3G4200D_ADDRESS_SA0_LOW);
00016     if (i2c.tryReadByte(L3G_WHO_AM_I) == 0xD3) return;
00017     i2c.addressSet(L3G4200D_ADDRESS_SA0_HIGH);
00018     if (i2c.tryReadByte(L3G_WHO_AM_I) == 0xD3) return;
00019     i2c.addressSet(L3GD20_ADDRESS_SA0_LOW);
00020     if (i2c.tryReadByte(L3G_WHO_AM_I) == 0xD4) return;
00021     i2c.addressSet(L3GD20_ADDRESS_SA0_HIGH);
00022     if (i2c.tryReadByte(L3G_WHO_AM_I) == 0xD4) return;
00023
00024     throw std::runtime_error("Could not detect gyro.");
00025 }
00026
00027 // Turns on the gyro and places it in normal mode.
00028 void L3G::enable()
00029 {
00030     detectAddress();
00031
00032     writeReg(L3G_CTRL_REG1, 0b00001111); // Normal power mode, all axes enabled
00033     writeReg(L3G_CTRL_REG4, 0b00000000); // 250 dps full scale
00034 }
00035
00036 void L3G::writeReg(uint8_t reg, uint8_t value)
00037 {
00038     i2c.writeByte(reg, value);
00039 }
00040
00041 uint8_t L3G::readReg(uint8_t reg)
00042 {
00043     return i2c.readByte(reg);
00044 }
00045
00046 void L3G::read()
00047 {
00048     uint8_t block[6];
00049     i2c.readBlock(0x80 | L3G_OUT_X_L, sizeof(block), block);
00050
00051     g[0] = (int16_t)(block[1] << 8 | block[0]);
00052     g[1] = (int16_t)(block[3] << 8 | block[2]);
00053     g[2] = (int16_t)(block[5] << 8 | block[4]);
00054 }
```

9.53 src/LSM303.cpp File Reference

`#include "LSM303.h"` Include dependency graph for LSM303.cpp:



Defines

- #define MAG_ADDRESS (0x3C >> 1)
- #define ACC_ADDRESS_SA0_A_LOW (0x30 >> 1)
- #define ACC_ADDRESS_SA0_A_HIGH (0x32 >> 1)

9.53.1 Define Documentation

9.53.1.1 #define ACC_ADDRESS_SA0_A_HIGH (0x32 >> 1)

Definition at line 20 of file LSM303.cpp.

9.53.1.2 #define ACC_ADDRESS_SA0_A_LOW (0x30 >> 1)

Definition at line 19 of file LSM303.cpp.

9.53.1.3 #define MAG_ADDRESS (0x3C >> 1)

Definition at line 18 of file LSM303.cpp.

9.54 src/LSM303.cpp

```
00001 #include "LSM303.h"
00002
00003 /*
00004
00005 Relevant Pololu products:
00006
00007 #1250  LSM303DLH              SA0_A pulled to GND, accessible via.
00008 #1264  LSM303DLH + L3G4200D   SA0_A pulled to GND, accessible thru-hole.
00009 #1265  LSM303DLM + L3G4200D   SA0_A pulled to GND, accessible thru-hole.
00010 #1268  LSM303DLHC + L3GD20
00011 #1273  LSM303DLM              SA0_A pulled to GND, accessible via.
00012 #2124  LSM303DLHC
00013
00014 LSM303DLHC has no SA0_A line
00015
00016  */
00017
00018 #define MAG_ADDRESS            (0x3C >> 1)
00019 #define ACC_ADDRESS_SA0_A_LOW  (0x30 >> 1)
00020 #define ACC_ADDRESS_SA0_A_HIGH (0x32 >> 1)
00021
00022 LSM303::LSM303(const char * i2cDeviceName) :
00023   i2c_mag(i2cDeviceName), i2c_acc(i2cDeviceName)
00024 {
00025 }
00026
00027 uint8_t LSM303::readMagReg(uint8_t reg)
00028 {
00029     return i2c_mag.readByte(reg);
00030 }
00031
00032 uint8_t LSM303::readAccReg(uint8_t reg)
00033 {
00034     return i2c_acc.readByte(reg);
00035 }
00036
00037 void LSM303::writeMagReg(uint8_t reg, uint8_t value)
00038 {
00039     i2c_mag.writeByte(reg, value);
00040 }
00041
00042 void LSM303::writeAccReg(uint8_t reg, uint8_t value)
00043 {
00044     i2c_acc.writeByte(reg, value);
00045 }
00046
00047 // Turns on the LSM303's accelerometer and magnetometers and places them in
       normal
00048 // mode.
00049 void LSM303::enable(void)
00050 {
00051     i2c_mag.addressSet(MAG_ADDRESS);
00052
00053     // Detect the accelerometer address and device.
00054     i2c_acc.addressSet(ACC_ADDRESS_SA0_A_LOW);
00055     bool sa0_a_high = i2c_acc.tryReadByte(LSM303_CTRL_REG1_A) == -1;
00056     if (sa0_a_high)
00057     {
00058         i2c_acc.addressSet(ACC_ADDRESS_SA0_A_HIGH);
```

```
        // Only the DLHC should be responding on the high address.
        device = Device::LSM303DLHC;
    }
    else
    {
        // Only the DLM has a LSM303_WHO_AM_I_M register.
        device = i2c_mag.tryReadByte(LSM303_WHO_AM_I_M) == 0x3C ? Device::LSM303DLM : Device::LSM303DLH;
    }

    // Make sure to throw an exception if we don't have the right address.
    readAccReg(LSM303_CTRL_REG1_A);

    if (readMagReg(LSM303_WHO_AM_I_M) != 0x3C)
    {
        throw std::runtime_error("LSM303: Error getting \"Who Am I\" register.\n");
    }

    // Enable accelerometer.
    if (device == Device::LSM303DLHC)
    {
        writeAccReg(LSM303_CTRL_REG1_A, 0b01000111); // Normal power mode, all axes enabled, 50 Hz
        writeAccReg(LSM303_CTRL_REG4_A, 0b10001000); // 2 g full scale: FS = 00 on DLHC, high resolution output mode
    }
    else
    {
        writeAccReg(LSM303_CTRL_REG1_A, 0b00100111); // normal power mode, all axes enabled, 50 Hz
        writeAccReg(LSM303_CTRL_REG4_A, 0b10000000); // 2 g full scale: FS = 00 on DLH, DLM
    }

    // Enable magnetometer
    // Continuous conversion mode
    writeMagReg(LSM303_CRB_REG_M, 0b01000000);  // FS = +-1.9 gauss
    writeMagReg(LSM303_MR_REG_M, 0x00);
}

void LSM303::readAcc(void)
{
    uint8_t block[6];
    i2c_acc.readBlock(0x80 | LSM303_OUT_X_L_A, sizeof(block), block);

    a[0] = (int16_t)(block[0] | block[1] << 8) >> 4;
    a[1] = (int16_t)(block[2] | block[3] << 8) >> 4;
    a[2] = (int16_t)(block[4] | block[5] << 8) >> 4;
}

void LSM303::readMag(void)
{
    uint8_t block[6];
    i2c_mag.readBlock(0x80 | LSM303_OUT_X_H_M, sizeof(block), block);

    // DLM, DLHC: register address order is X,Z,Y with high bytes first
    m[0] = (int16_t)(block[1] | block[0] << 8);
    m[1] = (int16_t)(block[5] | block[4] << 8);
    m[2] = (int16_t)(block[3] | block[2] << 8);

    // TODO: handle DLH properly here (switch two components?)

}

// Reads all 6 channels of the LSM303 and stores them in the object variables
void LSM303::read(void)
{
    readAcc();
    readMag();
}
```

9.55 src/main.cpp File Reference

`#include <csignal>` `#include <cstdlib>` `#include <unistd.h>` `#include <iostream>` `#include <string>` `#include "tclap/CmdLine.h"` `#include "mainthread.h"` Include dependency graph for main.cpp:

Functions

- TCLAP::CmdLine cmd ("Program for the attitude determination and control of the USU simulation table",' ',"0.1")
- TCLAP::ValueArg< string > trajFile ("","trajfile","Input file for the trajectory the table should follow", false,"input.txt","filename")
- TCLAP::ValueArg< float > pgain ("","pgain","The P-Gain for the simple proportional speed controller", false, 1.0,"float")
- TCLAP::ValueArg< string > mode ("","mode", modeText, true, string(),"mode name")
- void endProgram (int s)
- int main (int argc, char **argv)

Variables

- const string modeText
- MainThread kalmanFilter (5, 20000,"/dev/i2c-2","/dev/i2c-3")

9.55.1 Function Documentation

9.55.1.1 TCLAP::CmdLine **cmd** ("Program for the attitude determination and control of the USU simulation table" , ' ' , "0.1")

9.55.1.2 void **endProgram** (int *s*)

Definition at line 35 of file main.cpp.

9.55.1.3 int **main** (int *argc,* char ** *argv*)

Definition at line 43 of file main.cpp.

9.55.1.4 TCLAP::ValueArg<string> **mode** ("" , "mode" , **modeText** , true , string() , "mode name")

9.55.1.5 TCLAP::ValueArg<float> **pgain** ("" , "pgain" , "The P-Gain for the simple proportional speed controller" , false , 1. *0,* "float")

9.55.1.6 TCLAP::ValueArg< string > trajFile ("" , "trajfile" , "Input file for the trajectory the table should follow" , false , "input.txt" , "filename")

9.55.2 Variable Documentation

9.55.2.1 MainThread kalmanFilter(5, 20000,"/dev/i2c-2","/dev/i2c-3")

9.55.2.2 const string modeText

Initial value:

```
string("Operation mode: \n\t") +
                                 string("- pololu: Collect data from Pololu IMU
      and print it in csv format\n\t") +
                                 string("- microstrain: Collect data from
      MicroStrain IMU and print it in csv format\n\t") +
                                 string("- collect: Collect data from both IMUs
      and print it in csv format\n\t") +
                                 string("- simpleControl: Run simple angular
      velocity control scheme")
```

Definition at line 14 of file main.cpp.

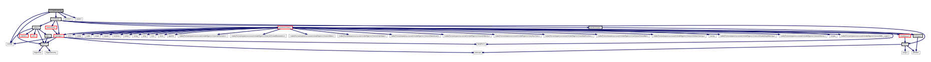
9.56 src/main.cpp

```
00001 #include<csignal>
00002 #include<cstdlib>
00003 #include<unistd.h>
00004
00005 #include <iostream>
00006 #include <string>
00007 using std::string;
00008
00009 #include "tclap/CmdLine.h"
00010 #include "mainthread.h"
00011 using namespace USU;
00012
00013 // Text to explain the different modes (more elegant way to split strings over
       several lines?)
00014 const string modeText = string("Operation mode: \n\t") +
00015                                  string("- pololu: Collect data from Pololu IMU
       and print it in csv format\n\t") +
00016                                  string("- microstrain: Collect data from
       MicroStrain IMU and print it in csv format\n\t") +
00017                                  string("- collect: Collect data from both IMUs
       and print it in csv format\n\t") +
00018                                  string("- simpleControl: Run simple angular
       velocity control scheme");
00019
00020 // Parse the command line arguments
00021 // Define possible arguments
00022 TCLAP::CmdLine cmd("Program for the attitude determination and control of the
       USU simulation table",' ', "0.1");
00023
00024 TCLAP::ValueArg<string> trajFile("", "trajfile", "Input file for the trajectory
       the table should follow", false, "input.txt", "filename");
00025 TCLAP::ValueArg<float> pgain("", "pgain", "The P-Gain for the simple
       proportional speed controller", false, 1.0, "float");
00026 TCLAP::ValueArg<string> mode("", "mode",  modeText , true, string(), "mode name
      ");
00027
00028 // Example for switching arg
00029 //TCLAP::SwitchArg stats("s", "stats", "Print statistics (number of spots,
       number of identified spots, ratio");
00030
```

```

MainThread kalmanFilter(5, 20000 , "/dev/i2c-2", "/dev/i2c-3");
//bool run = true;

void endProgram(int s)
{
    std::cerr << "MAIN: Got signal for termination" << std::endl;
    std::cerr << "MAIN: Stopping kalman filter thread..." << std::endl;
    kalmanFilter.stop();
}


int main(int argc, char **argv)
{
    // Register endProgram function as
    // signal handler for the kill signal (ctrl+c)
    struct sigaction sigIntHandler;
    sigIntHandler.sa_handler = endProgram;
    sigemptyset(&sigIntHandler.sa_mask);
    sigIntHandler.sa_flags = 0;

    sigaction(SIGINT, &sigIntHandler, NULL);

    try
    {
        // Register commandline options to parser
        cmd.add(trajFile);
        cmd.add(pgain);
        cmd.add(mode);

        cmd.parse(argc, argv);

        // Evaluate command line options
        if(mode.getValue() == "simpleControl")
        {
            kalmanFilter.initializeModeSimpleControl(trajFile.getValue(), pgain.getValue());
            kalmanFilter.setMode(MainThread::SimpleControl);
        }
        else if(mode.getValue() == "pololu")
        {
            kalmanFilter.setMode(MainThread::CollectPololuData);
        }
        else if(mode.getValue() == "microstrain")
        {
            kalmanFilter.setMode(MainThread::CollectMicroStrainData);
        }
        else if(mode.getValue() == "collect")
        {
            kalmanFilter.setMode(MainThread::CollectData);
        }
        else
        {
            throw std::runtime_error("MAIN: Unknown mode selected. Terminating");
            return 1;
        }

        kalmanFilter.start();

        if(kalmanFilter.join() )
        {
            std::cerr << "MAIN: Kalman filter thread joined" << std::endl;
            std::cerr << "MAIN: Terminating now..." << std::endl;
            return 0;
        }
        else
        {
            std::cerr << "MAIN: Joining Kalman filter thread failed" << std::endl;
            std::cerr << "MAIN: Terminating now..." << std::endl;
            return 1;
        }
    } catch (TCLAP::ArgException &e)  // catch any exceptions
    {
        std::cerr << "error: " << e.error() << " for arg " << e.argId() <<
    std::endl;
        return 1;
    }

}
```

9.57 src/mainthread.cpp File Reference

`#include <iostream>` `#include <sys/time.h>` `#include <unistd.h>` `#include "mainthread.h"` `#include "vector.h"` Include dependency graph for mainthread.cpp:



Functions

- int timeval_subtract (struct timeval ∗result, struct timeval ∗x, struct timeval ∗y)

9.57.1 Detailed Description

C++ class for the sensor fusion and stated estimated. Based on the PeriodicRtThread class.

Author

Jan Sommer Created on: Apr 20, 2013

Definition in file mainthread.cpp.

9.57.2 Function Documentation

9.57.2.1 int timeval_subtract (struct timeval ∗ *result,* struct timeval ∗ *x,* struct timeval ∗ *y*)

Definition at line 26 of file mainthread.cpp.

9.58 src/mainthread.cpp

```

#include<iostream>
using std::cout;
using std::endl;

#include <sys/time.h>
#include <unistd.h>


#include "mainthread.h"
#include "vector.h"
```

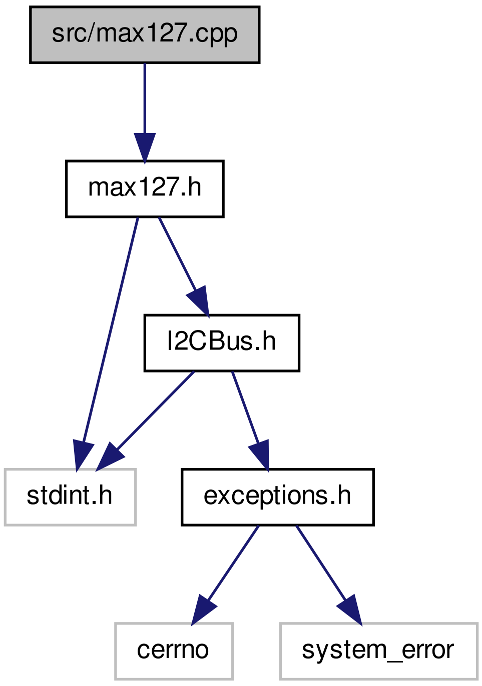
```cpp

using namespace USU;


int timeval_subtract (struct timeval * result, struct timeval * x, struct timeval * y)
{
    /* Perform the carry for the later subtraction by updating y. */
    if (x->tv_usec < y->tv_usec) {
        int nsec = (y->tv_usec - x->tv_usec) / 1000000 + 1;
        y->tv_usec -= 1000000 * nsec;
        y->tv_sec += nsec;
    }
    if (x->tv_usec - y->tv_usec > 1000000) {
        int nsec = (x->tv_usec - y->tv_usec) / 1000000;
        y->tv_usec += 1000000 * nsec;
        y->tv_sec -= nsec;
    }

    /* Compute the time remaining to wait.
          tv_usec is certainly positive. */
    result->tv_sec = x->tv_sec - y->tv_sec;
    result->tv_usec = x->tv_usec - y->tv_usec;

    /* Return 1 if result is negative. */
    return x->tv_sec < y->tv_sec;
}

MainThread::MainThread(int priority, unsigned int period_us, const char *i2cImu, const char *i2cMotor)
    :PeriodicRtThread(priority, period_us), mMode(CollectPololuData), mImu(i2cImu),
      mGX3(priority, "/dev/ttyO4"), mKeepRunning(false)
{
}

void MainThread::run()
{
    switch(mMode)
    {
    case SimpleControl:          runSimpleControl();
                                 break;
    case CollectPololuData:      runCollectPololu();
                                 break;
    case CollectMicroStrainData: runCollectMicroStrain();
                                 break;
    case CollectData:            runCollectBoth();
                                 break;
    }

    std::cerr << "KALMANFILTER: Terminating now..." << std::endl;
}

bool MainThread::getState()
{
    ScopedLock scLock(mStateLock);
    return mState;
}

void MainThread::initializeModeSimpleControl(std::string trajFilename, float pgain)
{
    std::ifstream inFile;
    inFile.open(trajFilename);
    if(!inFile.is_open())
        throw std::runtime_error("MotorController: Could not open input file");

    Command temp;
    while(true)
    {
        inFile >> temp.time;

        float x,y,z;
        inFile >> x >> y >> z;
        temp.angVel << x, y, z;

        if(inFile.eof())
            break;
        mCommandList.push_back(temp);
    }
    inFile.close();

    cout << "Read " << mCommandList.size() << " commands." << endl;

    mMotors.setPGain(pgain);
}

void MainThread::runSimpleControl()
{
    vector gyro;
    mKeepRunning = true;
    struct timeval start, now, elapsed;

    if(mCommandList.empty())
    {
        std::cerr << "Error: No command list loaded. Terminating";
        return;
    }

//    mImu.enable();

    std::vector<Command>::const_iterator commandIt = mCommandList.begin();
    mMotors.setSetValue(commandIt->angVel);
    int countdown = commandIt->time;


    gettimeofday(&start, NULL);
    unsigned lastTime = 0;
    waitPeriod();


    while(mKeepRunning)
    {
        gettimeofday(&now, NULL);

//        gyro = mImu.readGyro();

        // Run countdown
        timeval_subtract(&elapsed, &now, &start);
        unsigned time = elapsed.tv_sec * 1000 + elapsed.tv_usec / 1000; // in ms since start
        countdown -= time - lastTime;
        lastTime = time;

        // if countdown over execute next command from list
        if(countdown <=0)
        {
            commandIt++;
            // if at the end of the commandList start again from the beginning
            if(commandIt == mCommandList.end())
                commandIt = mCommandList.begin();

            countdown = commandIt->time;
            mMotors.setSetValue(commandIt->angVel);
        }

        // Alwasy use mutex, when changing state
         mMotors.controlFromGyro(gyro);

        waitPeriod();
    }

    gyro(2) = 1;
    mMotors.setSetValue(gyro);
    mMotors.controlFromGyro(gyro);
    waitPeriod();
    gyro(2) = 0;
    mMotors.setSetValue(gyro);
```

```cpp
    mMotors.controlFromGyro(gyro);

    std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
}

void MainThread::runCollectPololu()
{
    vector acc, mag, gyro;
    mKeepRunning = true;
    struct timeval start, now, elapsed;

    mImu.enable();

    gettimeofday(&start, NULL);
    waitPeriod();

    // Create an object to set the output format for the vectors
    Eigen::IOFormat csv(Eigen::StreamPrecision, Eigen::DontAlignCols, ", ", ", ", "");

    while(mKeepRunning)
    {
        gettimeofday(&now, NULL);

        acc  = mImu.readAcc();
        mag  = mImu.readMag();
        gyro = mImu.readGyro();

        timeval_subtract(&elapsed, &now, &start);
        unsigned time = elapsed.tv_sec * 1000 + elapsed.tv_usec / 1000; // in ms since start

        // print data
        cout << time << ",\t" << acc.format(csv) << ",\t" << mag.format(csv) << ",\t" << gyro.format(csv) << endl;

        waitPeriod();
    }

    std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
}

void MainThread::runCollectMicroStrain()
{

    mKeepRunning = true;

    mGX3.initialize();
    mGX3.start();
    packet_ptr lastState;
    while(mKeepRunning)
    {

        if(mGX3.isEmpty() == false)
        {
            int length = mGX3.size();
            while(length-->1)
            {
                mGX3.pop();
            }

            lastState = mGX3.front();
            mGX3.pop();

        }

        cout << (*lastState) << endl;
        waitPeriod();
    }

    std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
    std::cerr << "KALMANFILTER: Stopping Gx3-communicator..." << std::endl;
    mGX3.stop();
    if(mGX3.join() )
    {
        std::cerr << "KALMANFILTER: Gx3-communicator joined" << std::endl;
    }
    else
    {
        std::cerr << "KALMANFILTER: Joining Gx3-communicator failed" << std::endl;
    }
}

void MainThread::runCollectBoth()
{
    vector acc, mag, gyro;
    mKeepRunning = true;
    struct timeval start, now, elapsed;

    mImu.enable();
    mGX3.initialize();
    mGX3.start();

    gettimeofday(&start, NULL);
    waitPeriod();

    // Create an object to set the output format for the vectors
    Eigen::IOFormat csv(Eigen::StreamPrecision, Eigen::DontAlignCols, ", ", ", ");

    while(mKeepRunning)
    {
        gettimeofday(&now, NULL);

        acc  = mImu.readAcc();
        mag  = mImu.readMag();
        gyro = mImu.readGyro();

        timeval_subtract(&elapsed, &now, &start);
        unsigned time = elapsed.tv_sec * 1000 + elapsed.tv_usec / 1000; // in ms since start

        packet_ptr lastState;

        if(mGX3.isEmpty() == false)
        {
            int length = mGX3.size();
            while(length-->1)
            {
                mGX3.pop();
            }

            lastState = mGX3.front();
            mGX3.pop();
        }

        // print data
        cout << (*lastState) << "\t" << time << ",\t" << acc.format(csv) << ",\t" << mag.format(csv) << ",\t" << gyro.format(csv) << endl;

        waitPeriod();
    }

    std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
    std::cerr << "KALMANFILTER: Stopping Gx3-communicator..." << std::endl;
    mGX3.stop();
    if(mGX3.join() )
    {
        std::cerr << "KALMANFILTER: Gx3-communicator joined" << std::endl;
    }
    else
    {
        std::cerr << "KALMANFILTER: Joining Gx3-communicator failed" << std::endl;
    }
}
MainThread::Mode MainThread::getMode() const
```

```
{
    return mMode;
}

void MainThread::setMode(const Mode &value)
{
    mMode = value;
}

```

9.59 src/max127.cpp File Reference

`#include "max127.h"` Include dependency graph for max127.cpp:



9.59.1 Detailed Description

C++ class for the ADC Max127.

Author

Jan Sommer Created on: May 20, 2013

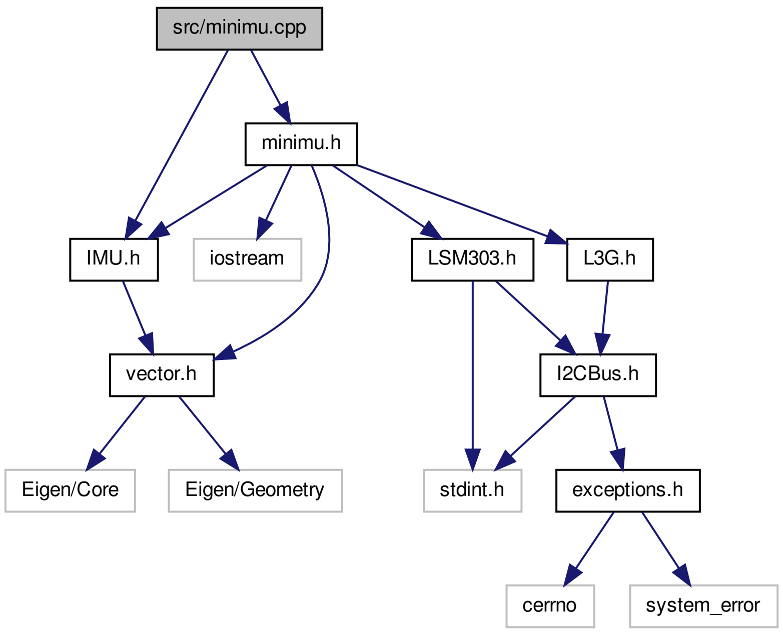
Definition in file max127.cpp.

9.60 src/max127.cpp

```
#include "max127.h"
using namespace USU;

Max127::Max127(const char *i2cdevice)
    :mI2c(i2cdevice)
{
    mI2c.addressSet(I2C_ADDRESS);
}

int16_t Max127::readRaw(uint8_t channel)
{
    // Compile the full control byte by setting the channel bits
    mI2c.writeByte(CONTROL_BYTE | (channel << SEL0) );

    uint16_t rawValue = mI2c.readWord();


    // From the read word use use the high byte as low byte and vice versa
    // Then move all bits 4 to the left because it is only a 12 bit number.
    return ( (int16_t) ( ( (rawValue & 0xFF00)>>8) | ( (rawValue & 0x00FF )<<8)
      ) >> 4 );


}

float Max127::readVoltage(unsigned int channel)
{
    // fullscale = +-5V --> resolution = 10V/4096
    const float scaleVoltage = 2.4414063e-3f;
    return (readRaw(channel) * scaleVoltage);
}
```

9.61 src/minimu.cpp File Reference

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

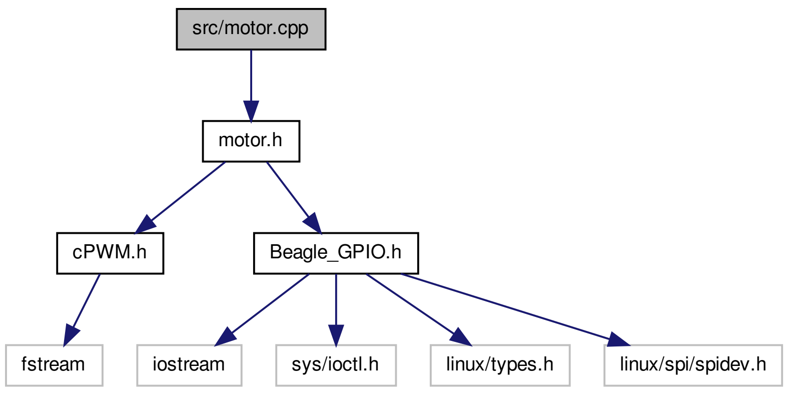


9.62 src/minimu.cpp

```
#include "minimu.h"
using namespace USU;
#include "IMU.h"

MinImu::MinImu(const char *i2cDeviceName)
    :compass(i2cDeviceName), gyro(i2cDeviceName)
{

}

void MinImu::enable()
{
    compass.enable();
    gyro.enable();
}

vector MinImu::readGyro()
{
    // At the full-scale=250 dps setting, the gyro datasheet says
    // we get 8.75 mdps/digit.
    const float gyro_scale = 0.00875; // in °/s

    gyro.read();
    IMU::raw_g = int_vector_from_ints(&gyro.g);
    return ( vector_from_ints(&gyro.g)  * gyro_scale);
}

vector MinImu::readAcc()
{
    // LSM303 accelerometer: At 2 g sensitivity, the datasheet says
    // we get 1 mg/digit.
    const float accel_scale = 0.0010;

    compass.readAcc();
    IMU::raw_a = int_vector_from_ints(&compass.a);
    return vector_from_ints(&compass.a) * accel_scale;
}

vector MinImu::readMag()
{
    compass.readMag();
    IMU::raw_m = int_vector_from_ints(&compass.m);

    return vector_from_ints(&compass.m);
}
```

9.63 src/motor.cpp File Reference

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



9.63.1 Detailed Description

Class to represent a motor

Author

Jan Sommer Created on: Apr 22, 2013

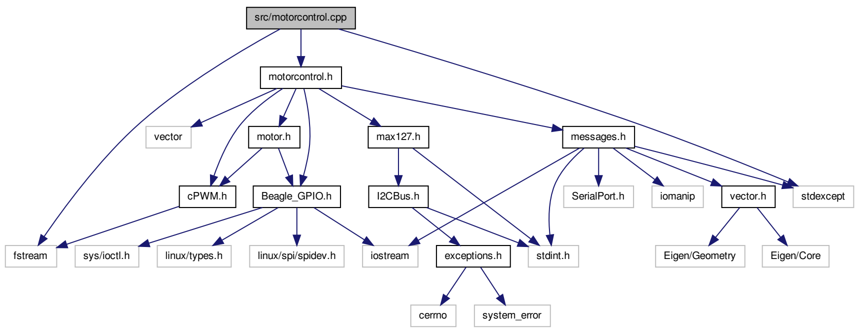
Definition in file motor.cpp.

9.64 src/motor.cpp

```

#include "motor.h"
using namespace USU;

Motor::Motor(Beagle_GPIO &beagleGpio, Beagle_GPIO::Pins clockwise,
Beagle_GPIO::Pins counterClockwise, cPWM &pwm, SetDutyCyle dutyCycle)
    :mBeagleGpio(beagleGpio), mClockwise(clockwise), mCounterClockwise(
counterClockwise),
      mPwm(pwm), mSetDutyCycle(dutyCycle), mSpeed(0)
{
    mBeagleGpio.configurePin(mClockwise, Beagle_GPIO::kOUTPUT);
    mBeagleGpio.enablePinInterrupts( mClockwise, false );
    mBeagleGpio.writePin(mClockwise, 0);

    mBeagleGpio.configurePin(mCounterClockwise, Beagle_GPIO::kOUTPUT);
    mBeagleGpio.enablePinInterrupts( mCounterClockwise, false );
    mBeagleGpio.writePin(mCounterClockwise, 0);

    setSpeed(mSpeed);
}

void Motor::setSpeed(int speed)
{
    if (speed > 0)
    {
        // Make sure speed <100
        speed = speed<100 ? speed : 99;
        mBeagleGpio.writePin(mClockwise, 1);
        mBeagleGpio.writePin(mCounterClockwise, 0);
        (mPwm.*mSetDutyCycle)(speed);
        mSpeed = speed;
    }
    else if (speed < 0)
    {
        // Make sure speed >-100
        speed = speed>-100 ? speed : -99;
        mBeagleGpio.writePin(mClockwise, 0);
        mBeagleGpio.writePin(mCounterClockwise, 1);
        (mPwm.*mSetDutyCycle)(-speed);
        mSpeed = speed;
    }
    else
    {
        mBeagleGpio.writePin(mClockwise, 0);
        mBeagleGpio.writePin(mCounterClockwise, 0);
        (mPwm.*mSetDutyCycle)(9);
        mSpeed = 0;
    }
}
```

9.65 src/motorcontrol.cpp File Reference

`#include <fstream>` `#include <stdexcept>` `#include "motorcontrol.h"` Include dependency graph for motorcontrol.cpp:



9.65.1 Detailed Description

C++ class for the calculation of the control response. Based on the PeriodicRtThread class.

Author

Jan Sommer Created on: Apr 22, 2013

Definition in file motorcontrol.cpp.

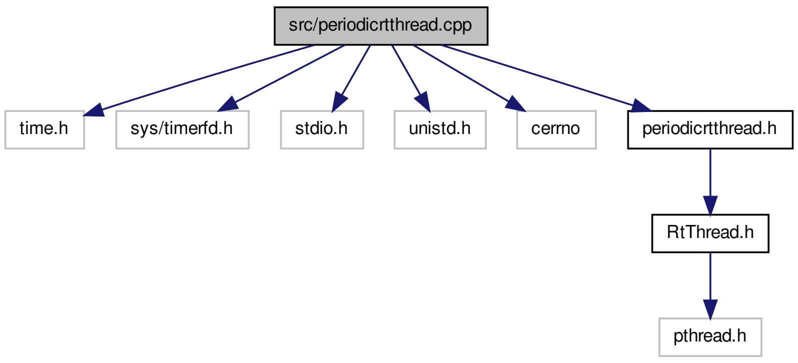
9.66 src/motorcontrol.cpp

```

#include <fstream>
#include <stdexcept>

#include "motorcontrol.h"
using namespace USU;

MotorControl::MotorControl(const char *i2cDevice)
    :mPwm1(1), mPwm2(2), mAnalog(i2cDevice), mPGain(1.0)
{
    // Initizalize the four motors
    mMotor[0] = new Motor(mBeagleGpio, Beagle_GPIO::P8_31,Beagle_GPIO::P8_29, mPwm1, &cPWM::DutyA_percent);
    mMotor[1] = new Motor(mBeagleGpio, Beagle_GPIO::P8_27,Beagle_GPIO::P8_25, mPwm1, &cPWM::DutyB_percent);
    mMotor[2] = new Motor(mBeagleGpio, Beagle_GPIO::P8_23,Beagle_GPIO::P8_21, mPwm2, &cPWM::DutyA_percent);
    mMotor[3] = new Motor(mBeagleGpio, Beagle_GPIO::P8_18,Beagle_GPIO::P8_17, mPwm2, &cPWM::DutyB_percent);
    mPwm1.Period_freq(100);
    mPwm2.Period_freq(100);
    mPwm1.RunA();
    mPwm1.RunB();
    mPwm2.RunA();
    mPwm2.RunB();
}

MotorControl::~MotorControl()
{
    mPwm1.StopA();
    mPwm1.StopB();
    mPwm2.StopA();
    mPwm2.StopB();
}

void MotorControl::calculateControlResponse(Quaternion state)
{
//    mMotor[0]->setSpeed(20);
}

void MotorControl::controlFromGyro(const Eigen::Vector3f & gyro)
{
//    float speeds[4];
//    float currents[4];
//    getAnalogs(speeds, currents);

//    int speeds_input[4];
//    getDutyCycles(speeds_input);

    int speed = (int) mSetValue(2);
    mMotor[0]->setSpeed(speed);
    mMotor[1]->setSpeed(speed);
    mMotor[2]->setSpeed(speed);
    mMotor[3]->setSpeed(speed);

    // mPGain I already have this, do the math to go from speeds to rpms (or
 rad/s), then Eigen::Vector3f err = (gyro - mSetValue)
    // From err to 4 * pwms

}

void MotorControl::setMotor(int motor, int dutyCycle)
{
    mMotor[motor]->setSpeed(dutyCycle);
}

void MotorControl::getAnalog(int motor, float& aOut1, float& aOut2)
{
    aOut1 =  mAnalog.readVoltage(motor*2);
    aOut2 =  mAnalog.readVoltage(motor*2 + 1);
}

void MotorControl::getAnalogs(float *aOut1, float *aOut2)
{
    aOut1[0] =  mAnalog.readVoltage(0);
    aOut2[0] =  mAnalog.readVoltage(1);
    aOut1[1] =  mAnalog.readVoltage(2);
    aOut2[1] =  mAnalog.readVoltage(3);
    aOut1[2] =  mAnalog.readVoltage(4);
    aOut2[2] =  mAnalog.readVoltage(5);
    aOut1[3] =  mAnalog.readVoltage(6);
    aOut2[3] =  mAnalog.readVoltage(7);
}

void MotorControl::getDutyCycles(int *dc)
{
    dc[0] = mMotor[0]->getSpeed();
    dc[1] = mMotor[1]->getSpeed();
    dc[2] = mMotor[2]->getSpeed();
    dc[3] = mMotor[3]->getSpeed();
}
float MotorControl::getPGain() const
{
    return mPGain;
}

void MotorControl::setPGain(float value)
```

```
{
    mPGain = value;
}
Eigen::Vector3f MotorControl::getSetValue() const
{
    return mSetValue;
}

void MotorControl::setSetValue(const Eigen::Vector3f value)
{
    mSetValue = value;
}


```

9.67 src/periodicrtthread.cpp File Reference

`#include <time.h>` `#include <sys/timerfd.h>` `#include <stdio.h>` `#include <unistd.h>` `#include <cerrno>` `#include "periodicrtthread.h"` Include dependency graph for periodicrtthread.cpp:



9.67.1 Detailed Description

Small C++ wrapper class to create a realtime scheduled pthread with periodic timer events.

Author

Jan Sommer Created on: Apr 10, 2013

Definition in file periodicrtthread.cpp.

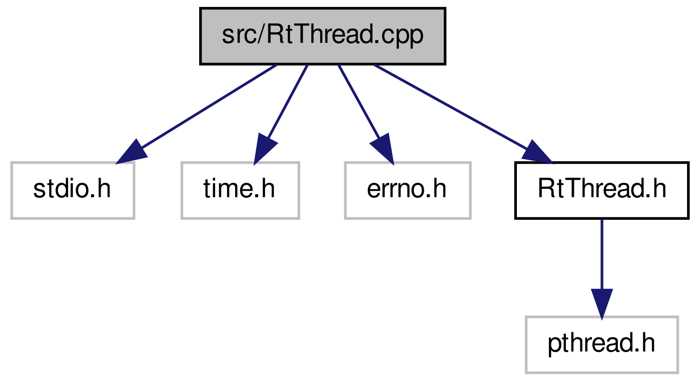
9.68 src/periodicrtthread.cpp

```

```

```
#include <time.h>
#include <sys/timerfd.h>
#include <stdio.h>
#include <unistd.h>
#include <cerrno>

#include "periodicrtthread.h"
using namespace USU;


PeriodicRtThread::PeriodicRtThread(int priority, unsigned int period_us)
    :RtThread(priority), mMissedWakeUps(0), mPeriod_us(period_us)
{
    makeThreadPeriodic();
}


void PeriodicRtThread::makeThreadPeriodic()
{
    int ret;
    unsigned int ns;
    unsigned int sec;
    struct itimerspec itval;
    /* Create the timer */
    if ( (mTimerFd = timerfd_create (CLOCK_MONOTONIC, 0)) == -1)
    {
        perror("timer_create ");
        return;
    }

    /* Make the timer periodic */
    sec = mPeriod_us/1000000;
    ns = (mPeriod_us - (sec * 1000000)) * 1000;
    itval.it_interval.tv_sec = sec;
    itval.it_interval.tv_nsec = ns;
    itval.it_value.tv_sec = sec;
    itval.it_value.tv_nsec = ns;
    if ( (ret = timerfd_settime(mTimerFd, 0, &itval, NULL)) != 0)
    {
        perror("timerfd_settime ");
        return;
    }
}

void PeriodicRtThread::waitPeriod()
{
    unsigned long long missed;
    int ret;

    /* Wait for the next timer event. If we have missed any the
           number is written to "missed"
       While loop to continue the wait if interrupted by a signal.
    */
    while( (ret = read (mTimerFd, &missed, sizeof (missed)) ) == -1)
    {
        perror("timer read ");
        return;
    }

    /* "missed" should always be >= 1, but just to be sure, check it is not 0
 anyway */
    if (missed > 0)
        mMissedWakeUps += (missed - 1);
}
```

9.69 src/RtThread.cpp File Reference

`#include <stdio.h>` `#include <time.h>` `#include <errno.h>` × `#include "RtThread.h"` Include dependency graph for RtThread.cpp:



9.69.1 Detailed Description

Small C++ wrapper class to create a realtime scheduled pthread

Author

Jan Sommer Created on: Apr 10, 2013

Definition in file RtThread.cpp.

9.70 src/RtThread.cpp

```
#include <stdio.h>
#include <time.h>
#include <errno.h>

#include "RtThread.h"
using namespace USU;



RtThread::RtThread(int priority):
    mPriority(priority), mId(-1), mStarted(false)
{
    int ret;
    if ( (ret = pthread_attr_init(&mAttr)) != 0)
    {
        perror("phtread_attr_init ");
        throw "Error";
    }
    // Set scheduler to (realtime) FIFO
    if ( (ret = pthread_attr_setschedpolicy(&mAttr, SCHED_FIFO)) != 0)
    {
        perror("pthread_attr_setschedpolicy");
        throw "Error";
    }

    // Change priority for the thread to mPriority
    struct sched_param param;
    if ( (ret = pthread_attr_getschedparam(&mAttr, &param)) != 0)
    {
        perror("pthread_attr_getschedparam");
        throw "Error";
    }
    param.__sched_priority = mPriority;

    if ( (ret = pthread_attr_setschedparam(&mAttr, &param)) != 0)
    {
        perror("pthread_attr_setschedparam");
        throw "Error";
    }

    /*
     * Set inherit scheduler attribut to PTHREAD_EXPLICIT_SCHED
     * otherwise the schedule attributes in mAttr will be ignored
     * and the same settings as the main thread will be inherited.
     */
    if ( (ret = pthread_attr_setinheritsched(&mAttr, PTHREAD_EXPLICIT_SCHED))
!= 0)
    {
        perror("pthread_attr_setinheritsched ");
        throw "Error";
    }
}

RtThread::~RtThread()
{
    /*
     * Make sure that the thread terminated properly
     * before deleting the instance
     */
    this->join();
    int ret;
    if ( (ret = pthread_attr_destroy(&mAttr)) != 0)
    {
        perror("pthread_attr_destroy");
        throw "Error";
    }
}

inline
pthread_t RtThread::getThreadId() const
{
    return mId;
}

inline
int RtThread::getPriority() const
{
    return mPriority;
}

void RtThread::start(void *arg)
{
    int ret;
    mArgs = arg;
    /*
    * Since pthread_create is a C library function, the 3rd argument is
    * a global function that will be executed by the thread. In C++, we
    * emulate the global function using the static member function that
    * is called exec. The 4th argument is the actual argument passed to
    * the function exec. Here we use this pointer, which is an instance
    * of the Thread class.
    */

    if ( (ret = pthread_create(&mId, &mAttr, &RtThread::exec, this)) !=0)
    {
        perror("thread_create ");
        throw "Error";
    }
    mStarted = true;
}

bool RtThread::join(int timeout_ms)
{
    //Allow the thread to wait for the termination status
    if (mStarted)
    {
        if(timeout_ms == 0)
        {
            if (pthread_join(mId, NULL) != 0) return false;
        }
        else
        {
            struct timespec ts;
            if (clock_gettime(CLOCK_REALTIME, &ts) == -1)
            {
                perror("clock_gettime ");
                throw "Error";
            }
            ts.tv_sec  += timeout_ms / 1000;
            ts.tv_nsec += timeout_ms * 1000000;

            int result =  pthread_timedjoin_np(mId, NULL, &ts);
            if (result == ETIMEDOUT)
                return false;
        }
        return true;
    }

    return false;
}

// Function which is actually executed by the thread
void * RtThread::exec(void *thr)
{
    reinterpret_cast<RtThread *> (thr)->run();
    return NULL;
}

```