

SimTab

Generated by Doxygen 1.7.6.1

Tue Oct 22 2013 21:14:11

Contents

1	Module Index	1
1.1	Modules	1
2	Namespace Index	1
2.1	Namespace List	1
3	Class Index	1
3.1	Class Hierarchy	1
4	Class Index	2
4.1	Class List	2
5	File Index	4
5.1	File List	4
6	Module Documentation	6
6.1	Utility classes for threading with pthread	6
6.1.1	Detailed Description	6
6.2	Classes related to communication with Pololu MinIMU	7
6.2.1	Detailed Description	7
6.3	Classes related to communication with MicroStrain 3DM-GX3	8
6.3.1	Detailed Description	8
6.3.2	Typedef Documentation	9
6.4	Classes related to controlling the motors	10
6.4.1	Detailed Description	10
7	Namespace Documentation	11
7.1	USU Namespace Reference	11
7.1.1	Detailed Description	13
7.1.2	Variable Documentation	13
8	Class Documentation	17
8.1	USU::AccAngMag Class Reference	17
8.1.1	Detailed Description	19
8.1.2	Member Enumeration Documentation	19

8.1.3	Constructor & Destructor Documentation	19
8.1.4	Member Function Documentation	19
8.1.5	Member Data Documentation	20
8.2	USU::AccAngMagOrientationMat Class Reference	21
8.2.1	Detailed Description	22
8.2.2	Member Enumeration Documentation	22
8.2.3	Constructor & Destructor Documentation	22
8.2.4	Member Function Documentation	23
8.2.5	Member Data Documentation	23
8.3	Beagle_GPIO Class Reference	24
8.3.1	Detailed Description	25
8.3.2	Member Enumeration Documentation	25
8.3.3	Constructor & Destructor Documentation	29
8.3.4	Member Function Documentation	29
8.3.5	Member Data Documentation	30
8.4	cPWM Class Reference	32
8.4.1	Detailed Description	32
8.4.2	Member Enumeration Documentation	33
8.4.3	Constructor & Destructor Documentation	33
8.4.4	Member Function Documentation	33
8.5	USU::GX3Command Class Reference	36
8.5.1	Detailed Description	36
8.5.2	Member Function Documentation	36
8.6	USU::GX3Communicator Class Reference	37
8.6.1	Detailed Description	38
8.6.2	Constructor & Destructor Documentation	39
8.6.3	Member Function Documentation	39
8.7	USU::GX3Packet Class Reference	40
8.7.1	Detailed Description	41
8.7.2	Member Function Documentation	42
8.8	I2CBus Class Reference	43
8.8.1	Detailed Description	44
8.8.2	Constructor & Destructor Documentation	44
8.8.3	Member Function Documentation	45

8.9	IMU Class Reference	47
8.9.1	Detailed Description	48
8.9.2	Member Function Documentation	48
8.9.3	Member Data Documentation	48
8.10	L3G Class Reference	49
8.10.1	Detailed Description	49
8.10.2	Constructor & Destructor Documentation	49
8.10.3	Member Function Documentation	50
8.10.4	Member Data Documentation	50
8.11	USU::Lock Class Reference	51
8.11.1	Detailed Description	51
8.11.2	Constructor & Destructor Documentation	51
8.11.3	Member Function Documentation	51
8.12	LSM303 Class Reference	52
8.12.1	Detailed Description	52
8.12.2	Constructor & Destructor Documentation	53
8.12.3	Member Function Documentation	53
8.12.4	Member Data Documentation	54
8.13	USU::MainThread Class Reference	55
8.13.1	Detailed Description	57
8.13.2	Member Enumeration Documentation	57
8.13.3	Constructor & Destructor Documentation	57
8.13.4	Member Function Documentation	57
8.14	USU::Max127 Class Reference	59
8.14.1	Detailed Description	59
8.14.2	Constructor & Destructor Documentation	59
8.14.3	Member Function Documentation	59
8.15	USU::MinImu Class Reference	60
8.15.1	Detailed Description	61
8.15.2	Constructor & Destructor Documentation	62
8.15.3	Member Function Documentation	62
8.15.4	Member Data Documentation	63
8.16	USU::Motor Class Reference	63
8.16.1	Detailed Description	63

8.16.2	Constructor & Destructor Documentation	64
8.16.3	Member Function Documentation	64
8.17	USU::MotorControl Class Reference	64
8.17.1	Detailed Description	65
8.17.2	Constructor & Destructor Documentation	65
8.17.3	Member Function Documentation	66
8.18	USU::PeriodicRtThread Class Reference	67
8.18.1	Detailed Description	69
8.18.2	Constructor & Destructor Documentation	69
8.18.3	Member Function Documentation	69
8.19	USU::Quaternion Class Reference	70
8.19.1	Detailed Description	71
8.19.2	Member Enumeration Documentation	72
8.19.3	Constructor & Destructor Documentation	72
8.19.4	Member Function Documentation	72
8.19.5	Member Data Documentation	73
8.20	USU::RawAccAng Class Reference	73
8.20.1	Detailed Description	74
8.20.2	Member Enumeration Documentation	75
8.20.3	Constructor & Destructor Documentation	75
8.20.4	Member Function Documentation	75
8.20.5	Member Data Documentation	76
8.21	USU::RtThread Class Reference	76
8.21.1	Detailed Description	77
8.21.2	Constructor & Destructor Documentation	77
8.21.3	Member Function Documentation	78
8.21.4	Member Data Documentation	79
8.22	USU::SamplingSettings Class Reference	80
8.22.1	Detailed Description	81
8.22.2	Member Enumeration Documentation	81
8.22.3	Constructor & Destructor Documentation	82
8.22.4	Member Function Documentation	83
8.22.5	Member Data Documentation	83
8.23	USU::ScopedLock Class Reference	84

8.23.1	Detailed Description	84
8.23.2	Constructor & Destructor Documentation	84
8.24	USU::Semaphore Class Reference	85
8.24.1	Detailed Description	85
8.24.2	Constructor & Destructor Documentation	85
8.24.3	Member Function Documentation	85
8.25	USU::SetContinuousMode Class Reference	86
8.25.1	Detailed Description	87
8.25.2	Member Enumeration Documentation	87
8.25.3	Constructor & Destructor Documentation	88
8.25.4	Member Function Documentation	88
8.25.5	Member Data Documentation	88
8.26	USU::SharedQueue< T > Class Template Reference	89
8.26.1	Detailed Description	89
8.26.2	Member Function Documentation	89
9	File Documentation	90
9.1	include/Beagle_GPIO.h File Reference	90
9.1.1	Define Documentation	92
9.2	include/Beagle_GPIO.h	92
9.3	include/cPWM.h File Reference	95
9.3.1	Detailed Description	97
9.3.2	Define Documentation	97
9.4	include/cPWM.h	98
9.5	include/doxygen.h File Reference	99
9.6	include/doxygen.h	99
9.7	include/exceptions.h File Reference	99
9.8	include/exceptions.h	100
9.9	include/gx3communicator.h File Reference	100
9.9.1	Detailed Description	101
9.10	include/gx3communicator.h	102
9.11	include/I2CBus.h File Reference	103
9.12	include/I2CBus.h	104
9.13	include/IMU.h File Reference	105

9.14	include/IMU.h	106
9.15	include/L3G.h File Reference	107
9.15.1	Define Documentation	109
9.16	include/L3G.h	111
9.17	include/Lock.h File Reference	112
9.17.1	Detailed Description	113
9.18	include/Lock.h	114
9.19	include/LSM303.h File Reference	115
9.19.1	Define Documentation	117
9.20	include/LSM303.h	122
9.21	include/mainthread.h File Reference	124
9.21.1	Detailed Description	124
9.22	include/mainthread.h	125
9.23	include/max127.h File Reference	126
9.23.1	Detailed Description	127
9.24	include/max127.h	128
9.25	include/messages.h File Reference	128
9.25.1	Detailed Description	130
9.26	include/messages.h	131
9.27	include/minimu.h File Reference	136
9.27.1	Detailed Description	138
9.28	include/minimu.h	138
9.29	include/motor.h File Reference	139
9.29.1	Detailed Description	140
9.29.2	Typedef Documentation	140
9.30	include/motor.h	140
9.31	include/motorcontrol.h File Reference	141
9.31.1	Detailed Description	142
9.32	include/motorcontrol.h	142
9.33	include/periodicrtthread.h File Reference	143
9.33.1	Detailed Description	144
9.34	include/periodicrtthread.h	144
9.35	include/RtThread.h File Reference	145
9.35.1	Detailed Description	146

9.36	include/RtThread.h	146
9.37	include/semaphore.h File Reference	147
9.37.1	Detailed Description	148
9.38	include/semaphore.h	148
9.39	include/sharedqueue.h File Reference	150
9.39.1	Detailed Description	151
9.40	include/sharedqueue.h	151
9.41	include/vector.h File Reference	152
9.41.1	Typedef Documentation	153
9.42	include/vector.h	153
9.43	src/Beagle_GPIO.cpp File Reference	154
9.44	src/Beagle_GPIO.cpp	154
9.45	src/cPWM.cpp File Reference	160
9.45.1	Detailed Description	160
9.46	src/cPWM.cpp	161
9.47	src/gx3communicator.cpp File Reference	163
9.47.1	Detailed Description	163
9.48	src/gx3communicator.cpp	164
9.49	src/I2CBus.cpp File Reference	166
9.50	src/I2CBus.cpp	166
9.51	src/L3G.cpp File Reference	168
9.51.1	Define Documentation	168
9.52	src/L3G.cpp	169
9.53	src/LSM303.cpp File Reference	170
9.53.1	Define Documentation	170
9.54	src/LSM303.cpp	171
9.55	src/main.cpp File Reference	173
9.55.1	Function Documentation	173
9.55.2	Variable Documentation	174
9.56	src/main.cpp	174
9.57	src/mainthread.cpp File Reference	176
9.57.1	Detailed Description	176
9.57.2	Function Documentation	176
9.58	src/mainthread.cpp	176

9.59	src/max127.cpp File Reference	181
9.59.1	Detailed Description	181
9.60	src/max127.cpp	182
9.61	src/minimu.cpp File Reference	183
9.62	src/minimu.cpp	183
9.63	src/motor.cpp File Reference	184
9.63.1	Detailed Description	184
9.64	src/motor.cpp	185
9.65	src/motorcontrol.cpp File Reference	186
9.65.1	Detailed Description	186
9.66	src/motorcontrol.cpp	186
9.67	src/periodicrthread.cpp File Reference	188
9.67.1	Detailed Description	188
9.68	src/periodicrthread.cpp	188
9.69	src/RtThread.cpp File Reference	190
9.69.1	Detailed Description	190
9.70	src/RtThread.cpp	190

1 Module Index

1.1 Modules

Here is a list of all modules:

Utility 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:

USU

TODO: Make some proper exceptions	11
--	-----------

3 Class Index

3.1 Class Hierarchy

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

Beagle_GPIO	24
cPWM	32
USU::GX3Command	36
USU::SamplingSettings	80
USU::SetCountinuousMode	86
USU::GX3Packet	40
USU::AccAngMag	17
USU::AccAngMagOrientationMat	21
USU::Quaternion	70
USU::RawAccAng	73
I2CBus	43
IMU	47
USU::MinImu	60
L3G	49
USU::Lock	51
LSM303	52
USU::Max127	59
USU::Motor	63
USU::MotorControl	64
USU::RtThread	76
USU::GX3Communicator	37

USU::PeriodicRtThread	67
USU::MainThread	55
USU::ScopedLock	84
USU::Semaphore	85
USU::SharedQueue< T >	89

4 Class Index

4.1 Class List

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

USU::AccAngMag Representation for receiving acceleration, angular rate and magnetometer packets	17
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:	21
Beagle_GPIO Wrapper class to access the GPIOs of the BeagleBone	24
cPWM Wrapper class to access the PWM-devices of the BeagleBone	32
USU::GX3Command Base class for commands send to the 3DM-GX3-25	36
USU::GX3Communicator	37
USU::GX3Packet Abstract base class for received packets	40
I2CBus Wrapper class for I2C-bus communication	43
IMU Virtual base class for IMU	47
L3G Class to manage the communication to the L3G gyroscope via the I2C-bus	49

USU::Lock	51
Wrapper class for pthread mutexes	
LSM303	52
Class to manage communication to the LSM303 compass via the I2-C-bus	
USU::MainThread	55
Represents the Periodic Thread class for state estimation	
USU::Max127	59
Class representing the MAX127 ADC	
USU::MinImu	60
Class to manage the communication to the Pololu MinIMU9	
USU::Motor	63
Class which represents a motor	
USU::MotorControl	64
Represents the class for motor control	
USU::PeriodicRtThread	67
TODO: Make some proper exceptions	
USU::Quaternion	70
Representation for receiving the Quaternion representation from the IMU	
USU::RawAccAng	73
Representation for receiving (raw) acceleration & angular rate packets	
USU::RtThread	76
Abstract wrapper class for the pthread library with RT-priority	
USU::SamplingSettings	80
Represents the "Sampling Settings" command	
USU::ScopedLock	84
Provides a helper class for Scoped Mutexes	
USU::Semaphore	85
Wrapper class for semaphores	
USU::SetContinuousMode	86
Represents the "Set continuous mode" command	
USU::SharedQueue< T >	89
Wrapper class to make std::queue thread safe	

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

include/Beagle_GPIO.h	90
include/cPWM.h	95
include/doxygen.h	99
include/exceptions.h	99
include/gx3communicator.h	100
include/I2CBus.h	103
include/IMU.h	105
include/L3G.h	107
include/Lock.h	112
include/LSM303.h	115
include/mainthread.h	124
include/max127.h	126
include/messages.h	128
include/minimu.h	136
include/motor.h	139
include/motorcontrol.h	141
include/periodicrthread.h	143
include/RtThread.h	145
include/semaphore.h	147
include/sharedqueue.h	150
include/vector.h	152
src/Beagle_GPIO.cpp	154
src/cPWM.cpp	160
src/gx3communicator.cpp	163

src/I2CBus.cpp	166
src/L3G.cpp	168
src/LSM303.cpp	170
src/main.cpp	173
src/mainthread.cpp	176
src/max127.cpp	181
src/minimu.cpp	183
src/motor.cpp	184
src/motorcontrol.cpp	186
src/periodicrtthread.cpp	188
src/RtThread.cpp	190

6 Module Documentation

6.1 Utility classes for threading with pthread

Collaboration diagram for Utility classes for threading with pthread:



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

- class [I2C Bus](#)
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

- 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::SetContinuousMode](#)
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

- 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 [SetContinuousMode](#)
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 [SELO](#) = 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 [I2C Bus](#).

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](#).

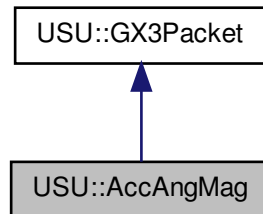
8 Class Documentation

8.1 USU::AccAngMag Class Reference

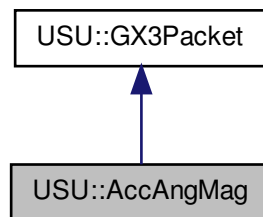
Representation for receiving acceleration, angular rate and magnetometer packets.

```
#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

<i>os</i>	
-----------	--

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

<i>serialPort</i>	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:

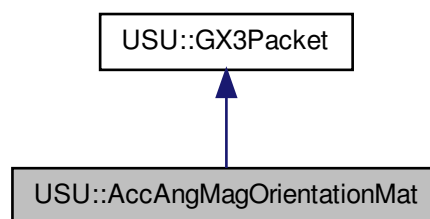
- include/[messages.h](#)

8.2 USU::AccAngMagOrientationMat Class Reference

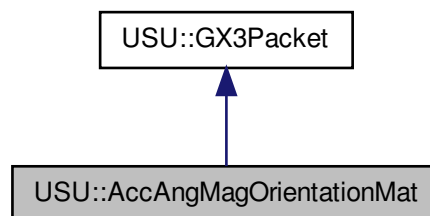
Representation for packets containing the 3 sensor vectors and orientation matrix This class can be used with the commands which return 3 Vectors and a 3x3 Matrix. The units are:

```
#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

<i>os</i>	
-----------	--

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

<i>serialPort</i>	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

<i>in</i>	<i>id</i>	id of the PWMss to be initialized. 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 higher 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	<i>nanoseconds</i> , :	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	<i>percent</i> ,:	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	<i>nanoseconds</i> , :	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	<i>percent</i> ,:	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	<i>freq_Hz</i> ,:	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	<i>nanoseconds</i>	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	<i>polarity</i>	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	<i>polarity</i>	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:

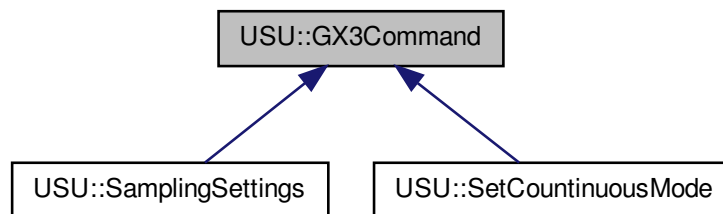
- [include/cPWM.h](#)
- [src/cPWM.cpp](#)

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::SetContinuousMode](#).

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

Implemented in [USU::SamplingSettings](#), and [USU::SetContinuousMode](#).

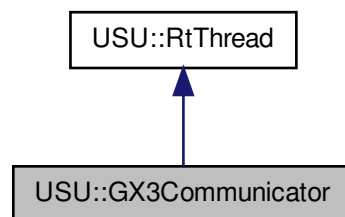
The documentation for this class was generated from the following file:

- [include/messages.h](#)

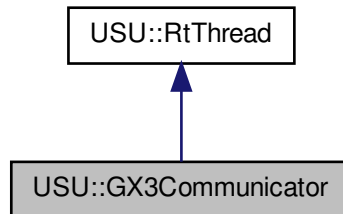
8.6 USU::GX3Communicator Class Reference

```
#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

<i>priority</i>	Priority of the pthread (1..99)
<i>serialDevice</i>	Name of the serial device
<i>baudRate</i>	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:

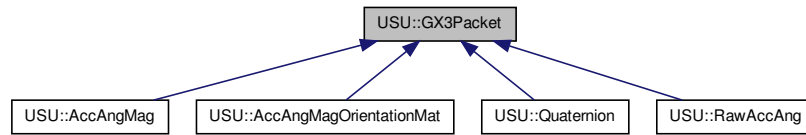
- [include/gx3communicator.h](#)
- [src/gx3communicator.cpp](#)

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 successive 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 matrices 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

<i>buffer</i>	pointer to the byte array
<i>length</i>	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 endianness of the host system and the 3DM match. The endianness of the sent floats can be set with the [SamplingSettings](#) command.

Parameters

<i>buffer</i>	Pointer to the byte array
<i>mat</i>	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

<i>buffer</i>	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 successive bytes.

NOTE: Make sure that the endianness of the host system and the 3DM match. The endianness of the sent floats can be set with the [SamplingSettings](#) command.

Parameters

<i>buffer</i>	Pointer to the byte array
---------------	---------------------------

Returns

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

<i>os</i>	
-----------	--

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

<i>serialPort</i>	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 I2C Bus 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

<i>deviceName</i>	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

<i>address</i>	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

<i>command</i>	Register to start reading from
<i>size</i>	Number of bytes to read
<i>data</i>	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

<i>command</i>	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

<i>command</i>	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

<i>command</i>	
----------------	--

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

<i>command</i>	Register to write the byte to
<i>data</i>	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

<i>data</i>	Byte of data which will be written directly to the device set with addressSet()
-------------	---

Definition at line 40 of file [I2CBus.cpp](#).

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

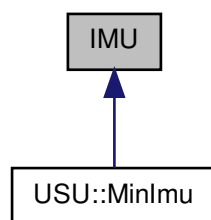
- [include/I2CBus.h](#)
- [src/I2CBus.cpp](#)

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

<i>i2cDevice- Name</i>	
----------------------------	--

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

<i>reg</i>	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

<i>reg</i>	Register address to write to
<i>value</i>	Value to write to the register <code>reg</code>

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

<i>i2cDevice- Name</i>	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

<i>reg</i>	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

<i>reg</i>	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

<i>reg</i>	Register address to write to
<i>value</i>	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

<i>reg</i>	Register address to write to
<i>value</i>	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:

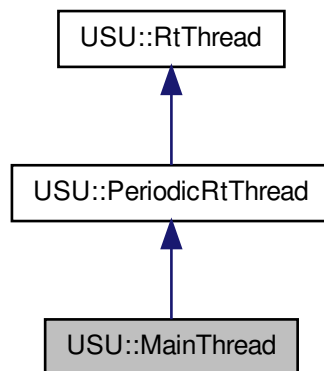
- [include/LSM303.h](#)
- [src/LSM303.cpp](#)

8.13 USU::MainThread Class Reference

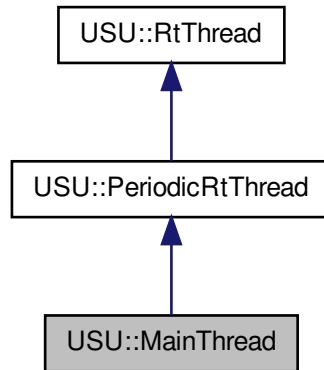
Represents the Periodic Thread class for state estimation.

```
#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, CollectData }

Public Member Functions

- **MainThread** (int priority, unsigned int period_us, const char *i2clmu, 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 * *i2clmu*, 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

<i>priority</i>	priority of the underlying periodic thread
<i>period_us</i>	period (in us) of the underlying periodic thread
<i>i2clmu</i>	name of the I2C-device for the IMU (e.g. /dev/i2c-1)
<i>i2cMotor</i>	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 accessing 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

<i>i2cdevice</i>	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

<i>channel</i>	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

<i>channel</i>	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:

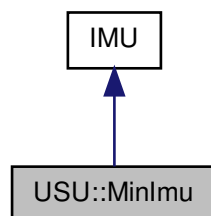
- [include/max127.h](#)
- [src/max127.cpp](#)

8.15 USU::MinImu Class Reference

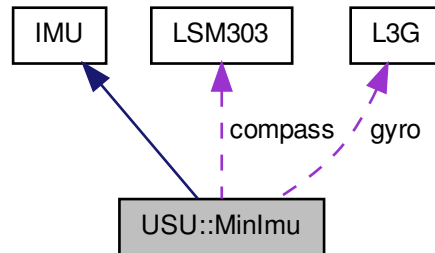
Class to manage the communication to the Pololu MiniIMU9.

```
#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

<i>i2cDevice- Name</i>	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

<i>beagleGpio</i>	Reference to a Beagle_GPIO object to set the pins
<i>clockwise</i>	First pin needed to set motor direction
<i>counter-Clockwise</i>	Second pin needed to set motor direction
<i>pwm</i>	Reference to the cPWM-object, which controls the PWM
<i>dutyCycle</i>	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

<i>speed</i>	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 [getAnalog](#)s (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 appropriate 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

<i>i2cDevice</i>	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

<i>state</i>	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

<i>gyro</i>	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

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

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

8.17.3.4 void MotorControl::getAnalog (float * aOut1, float * aOut2)

For testing: returns the Analog measurements of all motors.

Parameters

<i>aOut1</i>	Float array to store the first analog measurement of each motor
<i>aOut2</i>	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

<i>dc</i>	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

<i>motor</i>	which motor [0..3]
<i>dutyCycle</i>	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:

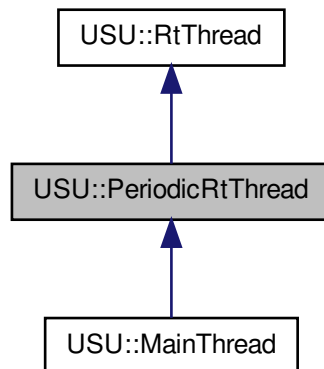
- include/[motorcontrol.h](#)
- src/[motorcontrol.cpp](#)

8.18 USU::PeriodicRtThread Class Reference

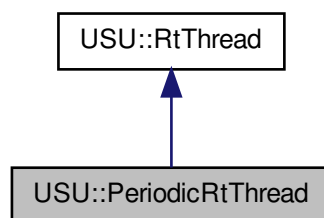
TODO: Make some proper exceptions.

```
#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

<i>priority</i>	the Priority of the Thread (Linux: 1..99)
<i>period_us</i>	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:

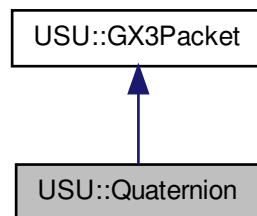
- include/[periodicrtthread.h](#)
- src/[periodicrtthread.cpp](#)

8.19 USU::Quaternion Class Reference

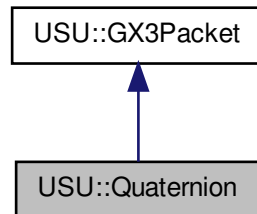
Representation for receiving the [Quaternion](#) representation from the [IMU](#).

```
#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

<i>os</i>	
-----------	--

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

<i>serialPort</i>	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:

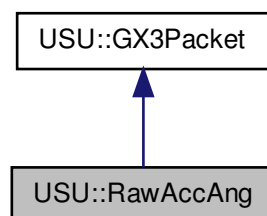
- include/[messages.h](#)

8.20 USU::RawAccAng Class Reference

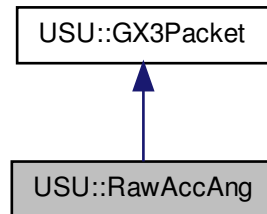
Representation for receiving (raw) acceleration & angular rate packets.

```
#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

<i>os</i>	
-----------	--

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

<i>serialPort</i>	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:

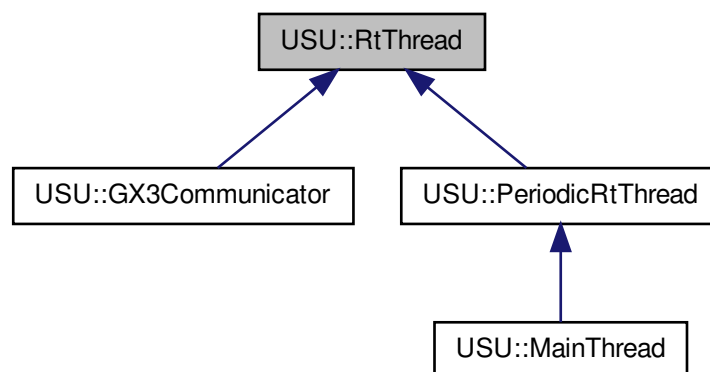
- include/[messages.h](#)

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_FIFO-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

<i>priority</i>	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

<i>thr</i>	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

<i>timeout_ms</i>	timeout in ms (optional). 0 means no timeout
-------------------	--

Returns

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

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

<i>args</i>	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](#)

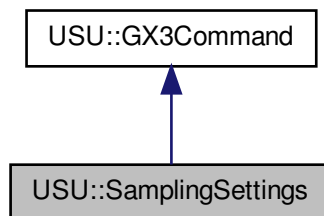
- [src/RtThread.cpp](#)

8.22 USU::SamplingSettings Class Reference

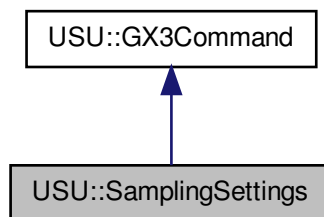
Represents the "Sampling Settings" command.

```
#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, [FlagSuppressNaN](#) = 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 gyroAccFilter=15, uint8_t magFilter=17, uint16_t upCompensation=10, uint16_t northCompensation=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

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

<i>buffer</i>	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

<i>lock</i>	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]

Tries 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]

Tries 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:

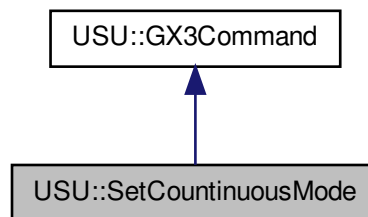
- include/[semaphore.h](#)

8.25 USU::SetContinuousMode Class Reference

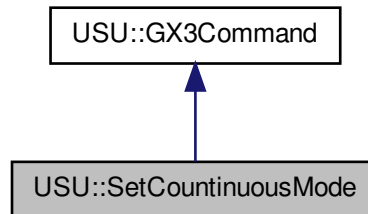
Represents the "Set continuous mode" command.

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

Inheritance diagram for USU::SetContinuousMode:



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::SetContinuousMode::SetContinuousMode (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

<i>Command-Byte</i>	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::SetContinuousMode::checkResponse (uint8_t * *buffer*) [inline, virtual]

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

Parameters

<i>buffer</i>	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::SetContinuousMode::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::SetContinuousMode::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 accessing 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

<i>newElement</i>	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:

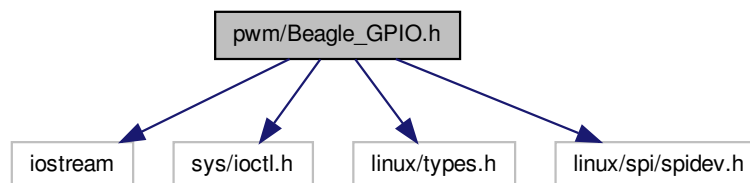
- include/[sharedqueue.h](#)

9 File Documentation

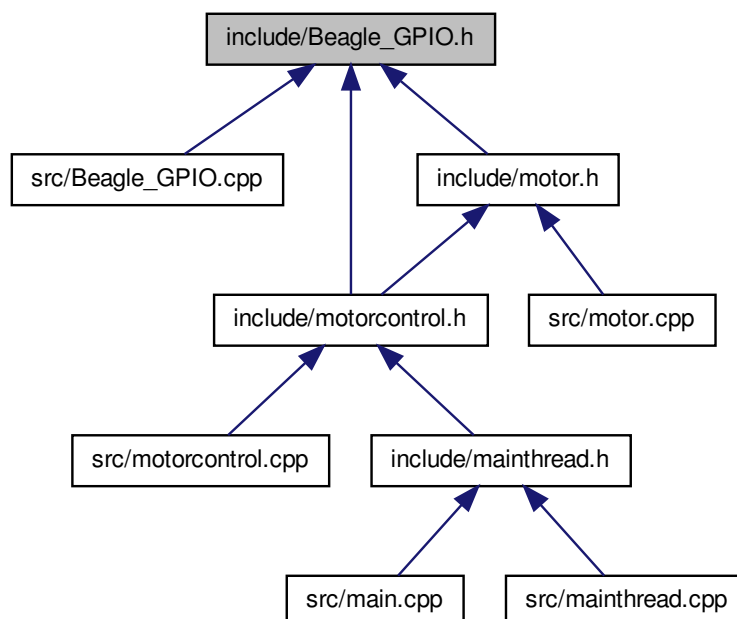
9.1 include/Beagle_GPIO.h File Reference

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

```
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

```
00001 /*****
00002  ** Beagle Bone GPIO Library **
00003  **                               **
00004  **      Francois Sugny          **
00005  **      01/07/12                **
00006  **                               **
00007  **      v1.0                    **
00008  *****/
00009
00010 //=====
00011 //=====
00012
00013 #ifndef beagle_gpio_hh
00014 #define beagle_gpio_hh
```

```

00015
00016 //=====
00017 //=====
00018
00019 #include <iostream>
00020 #include <sys/ioctl.h>
00021 #include <linux/types.h>
00022 #include <linux/spi/spidev.h>
00023
00024 //=====
00025 //=====
00026
00027 #define GPIO_ERROR(msg)      std::cout << "[GPIO] Error : " << msg <<
                                std::endl;
00028
00029 #define BEAGLE_GPIO_DEBUG
00030 #ifdef BEAGLE_GPIO_DEBUG
00031     #define GPIO_PRINT(msg)      std::cout << "[GPIO] : " << msg <<
                                std::endl;
00032     #define gp_assert( condition ) \
00033         if (!(condition)) \
00034         { \
00035             GPIO_ERROR( "Assert Failed in file \
                                \" << __FILE__ << \" on line \" << __LINE__ ); \
                                exit(0); \
                                } \
00036
00037
00038
00039 #else
00040     #define GPIO_PRINT(msg)
00041     #define gp_assert( condition )
00042 #endif
00043
00044
00045 //=====
00046 //=====
00047
00054 class Beagle_GPIO
00055 {
00056 public:
00057     // Return status
00058     typedef enum
00059     {
00060         kFail = 0,
00061         kSuccess = 1
00062     } Beagle_GPIO_Status;
00063
00064     // Beagle Bone GPIO Register Offsets
00065     enum
00066     {
00067         kREVISION = 0x0,
00068         kSYSCONFIG = 0x10,
00069         kIRQSTATUS_RAW_0 = 0x24,
00070         kIRQSTATUS_RAW_1 = 0x28,
00071         kIRQSTATUS_0 = 0x2C,
00072         kIRQSTATUS_1 = 0x30,
00073         kIRQSTATUS_SET_0 = 0x34,
00074         kIRQSTATUS_SET_1 = 0x38,
00075         kIRQSTATUS_CLR_0 = 0x3C,
00076         kIRQSTATUS_CLR_1 = 0x40,
00077         kIRQWAKEN_0 = 0x44,
00078         kIRQWAKEN_1 = 0x48,
00079         kSYSSTATUS = 0x114,
00080         kCTRL = 0x130,
00081         kOE = 0x134,
00082         kDATAIN = 0x138,
00083         kDATAOUT = 0x13C,
00084         kLEVELDETECT0 = 0x140,
00085         kLEVELDETECT1 = 0x144,
00086         kRISINGDETECT = 0x148,
00087         kFALLINGDETECT = 0x14C,
00088         kDEBOUNCEENABLE =

```

```

00089         0x150,
00090                                     kDEBOUNCINGTIME
00091         0x154,
00092                                     kCLEARDATAOUT
00093                                     kSETDATAOUT
00094                                     = 0x190,
00095                                     = 0x194
00096     } Beagle_GPIO_Registers;
00097
00098     // Input/Output pin mode
00099     typedef enum
00100     {
00101         kINPUT = 0,
00102         kOUTPUT = 1
00103     } Beagle_GPIO_Direction;
00104
00105     // GPIO Pins
00106     enum Pins
00107     {
00108         P8_1, P8_2, P8_3, P8_4, P8_5,
00109         P8_6, P8_7, P8_8, P8_9, P8_10,
00110         P8_11, P8_12, P8_13, P8_14, P8_15,
00111         P8_16, P8_17, P8_18, P8_19, P8_20,
00112         P8_21, P8_22, P8_23, P8_24, P8_25,
00113         P8_26, P8_27, P8_28, P8_29, P8_30,
00114         P8_31, P8_32, P8_33, P8_34, P8_35,
00115         P8_36, P8_37, P8_38, P8_39, P8_40,
00116         P8_41, P8_42, P8_43, P8_44, P8_45,
00117         P8_46,
00118         P9_1, P9_2, P9_3, P9_4, P9_5,
00119         P9_6, P9_7, P9_8, P9_9, P9_10,
00120         P9_11, P9_12, P9_13, P9_14, P9_15,
00121         P9_16, P9_17, P9_18, P9_19, P9_20,
00122         P9_21, P9_22, P9_23, P9_24, P9_25,
00123         P9_26, P9_27, P9_28, P9_29, P9_30,
00124         P9_31, P9_32, P9_33, P9_34, P9_35,
00125         P9_36, P9_37, P9_38, P9_39, P9_40,
00126         P9_41, P9_42, P9_43, P9_44, P9_45,
00127         P9_46
00128     } GPIO_Pins;
00129
00130     // IO Banks for GPIOs
00131     static const int GPIO_Pin_Bank[];
00132
00133     // Pin Id for GPIOs
00134     static const int GPIO_Pin_Id[];
00135
00136     // Pad Control Register
00137     static const unsigned long GPIO_Pad_Control[];
00138
00139     // Base address of Control Module Registers
00140     static const unsigned long GPIO_Control_Module_Registers;
00141
00142     // Base addresses of GPIO Modules
00143     static const unsigned long GPIO_Base[];
00144
00145 public:
00146     Beagle_GPIO();
00147     ~Beagle_GPIO();
00148
00149 public:
00150     // Configure pin as input/output
00151     Beagle_GPIO_Status configurePin( unsigned short _pin,
00152     Beagle_GPIO_Direction _direction );
00153
00154     // Enable/Disable interrupts for the pin
00155     Beagle_GPIO_Status enablePinInterrupts( unsigned short _pin,
00156     bool _enable );
00157
00158     // Write a value to a pin
00159     Beagle_GPIO_Status writePin( unsigned short _pin, unsigned char
00160     _value );
00161
00162     // Read a value from a pin
00163     unsigned char readPin( unsigned short _pin );
00164

```

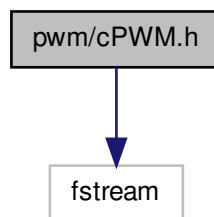
```

00158         // Open SPI Channel
00159         void openSPI( unsigned char _mode=0,
00160                      unsigned char _bits=8,
00161                      unsigned long _speed=4800000,
00162                      unsigned short _delay=0 );
00163
00164         // Close SPI Channel
00165         void closeSPI();
00166
00167         // Send SPI Buffer
00168         void sendSPIBuffer( unsigned long buffer, int size );
00169
00170         // Is this Module active ?
00171         bool isActive() { return m_active; }
00172
00173     private:
00174         bool                                m_active;
00175         int                                m_gpio_fd;
00176         unsigned long *                    m_controlModule;
00177         unsigned long *                    m_gpio[4];
00178
00179         int                                m_spi_fd;
00180         unsigned char *                    m_spi_buffer_rx;
00181         unsigned char                      m_spi_mode;
00182         unsigned char                      m_spi_bits;
00183         unsigned long                      m_spi_speed;
00184         unsigned short                    m_spi_delay;
00185
00186         struct spi_ioc_transfer m_spi_ioc_tr;
00187     };
00188
00189     //=====
00190     //=====
00191
00192 #endif
00193
00194     //=====
00195     //=====
00196

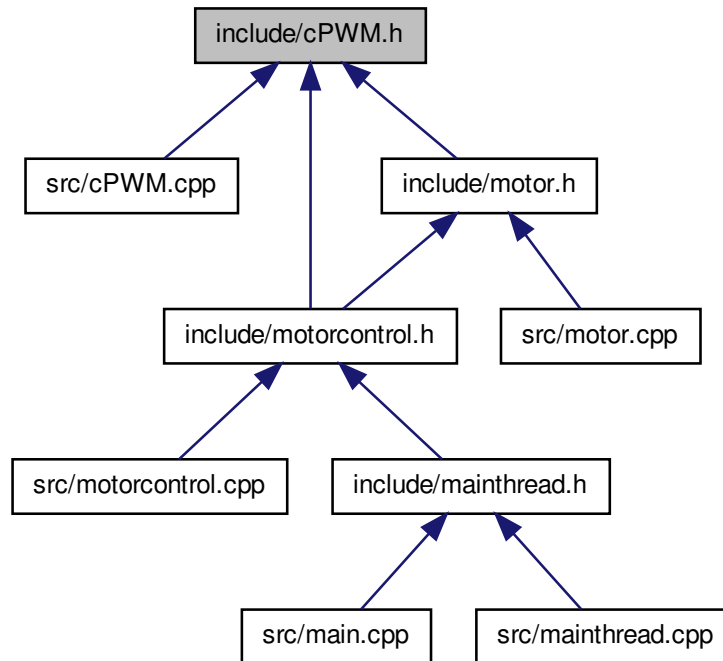
```

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

```

00001 // $Id$
00011 // $Log$
00012
00013 #ifndef CPWM_H_
00014 #define CPWM_H_
00015
00016 #include <fstream>
00017
00024 class cPWM {
00025
00026 public:
00027     enum Polarity
00028     {
00029         ActiveHigh,
00030         ActiveLow
00031     };
00032
00033 private:
00034     int id;
00035     unsigned int dutyA;
00036     unsigned int dutyB;
00037     unsigned int period;
00038     unsigned int freq_Hz;
00039     enum cPWM::Polarity polarityA;
00040     enum cPWM::Polarity polarityB;
00041     int runA;
00042     int runB;
00043     /*****
00044         *
00045         * sysfs tree:
00046         *
00047     ehrpwm.0:0
00048         duty_ns
00049         period_ns
00050         polarity
00051         request
00052         run
00053     ehrpwm.0:1 -> ../../devices/platform/omap/ehrpwm.0/pwm/ehrpwm.0:1
00054         duty_ns
00055         period_ns
00056         polarity
00057         request
00058         run
00059         *
00060     std::stringstream sysfs_file;
00061
00062     Define files to match sysfs tree:
00063         */
00064     ehrpwm."                #define SYSFS_EHRPWM_PREFIX "/sys/class/pwm/
00065
00066                             #define SYSFS_EHRPWM_SUFFIX_A ":0"
00067                             #define SYSFS_EHRPWM_SUFFIX_B ":1"
00068         #define SYSFS_EHRPWM_DUTY_NS "duty_ns"
00069         #define SYSFS_EHRPWM_DUTY_PERCENT "duty_percent"
00070         #define SYSFS_EHRPWM_PERIOD_NS "period_ns"
00071         #define SYSFS_EHRPWM_PERIOD_FREQ "period_freq"
00072                             #define SYSFS_EHRPWM_POLARITY "polarity"
00073                             #define SYSFS_EHRPWM_RUN "run"
00074                             #define SYSFS_EHRPWM_REQUEST "request"
00075
00076     std::ofstream sysfsfid_dutyA_ns;
00077     std::ofstream sysfsfid_dutyA_percent;
00078     std::ofstream sysfsfid_dutyB_ns;
00079     std::ofstream sysfsfid_dutyB_percent;
00080     std::ofstream sysfsfid_period_ns;
00081     std::ofstream sysfsfid_period_freq;
00082     std::ofstream sysfsfid_polarityA;
00083     std::ofstream sysfsfid_runA;
00084     std::ofstream sysfsfid_requestA;
00085     std::ofstream sysfsfid_polarityB;
00086     std::ofstream sysfsfid_runB;

```

```

00086                                     std::ofstream sysfsfid_requestB;
00087
00088 public:
00089
00090         cPWM(int id);
00091         virtual ~cPWM();
00092
00093         void DutyA_ns(unsigned int nanoseconds);
00094         void DutyA_percent(unsigned int percent); //TODO: check if floats are
possible
00095
00096         void DutyB_ns(unsigned int nanoseconds);
00097         void DutyB_percent(unsigned int percent); //TODO: check if floats are
possible
00098
00099         void Period_ns(unsigned int nanoseconds);
00100         void Period_freq(unsigned int freq_Hz);
00101
00102         void PolarityA(cPWM::Polarity polarity);
00103         void RunA();
00104         void StopA();
00105         void PolarityB(cPWM::Polarity polarity);
00106         void RunB();
00107         void StopB();
00108 };
00109
00110 #endif /* CPWM_H_ */

```

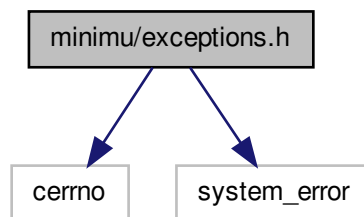
9.5 include/doxygen.h File Reference

9.6 include/doxygen.h

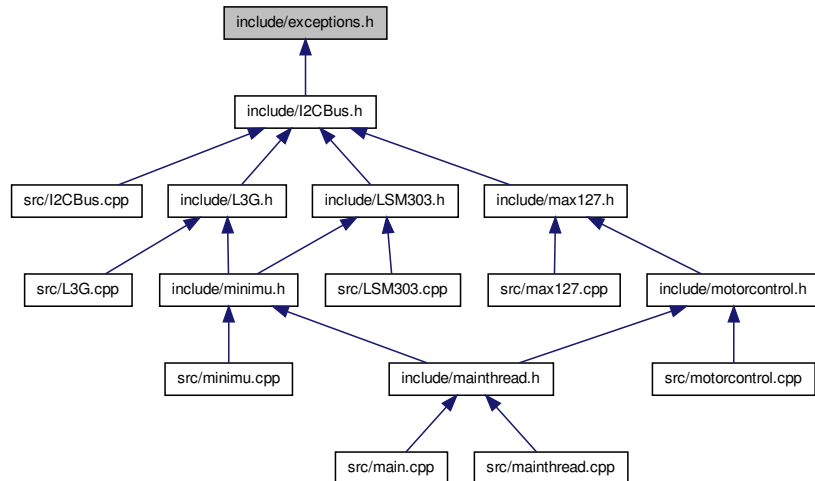
```
00001
```

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

```

00001 #ifndef _AHR5_EXCEPTIONS_H
00002 #define _AHR5_EXCEPTIONS_H
00003
00004 #include <cerrno>
00005 #include <system_error>
00006
00007 static inline std::system_error posix_error()
00008 {
00009     return std::system_error(errno, std::system_category());
00010 }
00011
00012 static inline std::system_error posix_error(const char * what)
00013 {
00014     return std::system_error(errno, std::system_category(), what);
00015 }
00016
00017 #endif

```

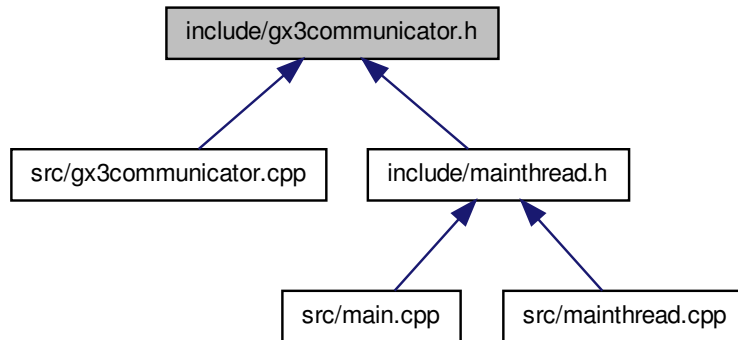
9.9 include/gx3communicator.h File Reference

```

#include <SerialPort.h> #include <memory> #include "Rt-
Thread.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

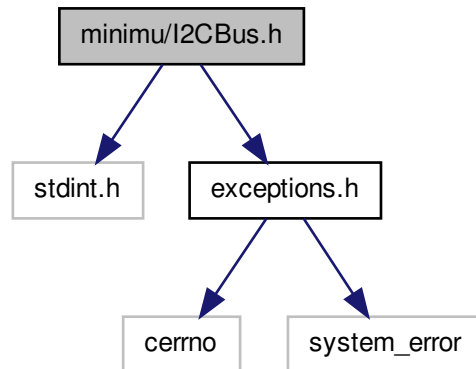
```

00001
00012 #ifndef GX3COMMUNICATOR_H
00013 #define GX3COMMUNICATOR_H
00014
00015 #include<SerialPort.h>
00016 #include<memory>
00017 #include "RtThread.h"
00018 #include "sharedqueue.h"
00019 #include "messages.h"
00020
00021 namespace USU
00022 {
00023
00044 typedef std::shared_ptr<GX3Packet> packet_ptr;
00045
00046 class GX3Communicator : public RtThread
00047 {
00048 public:
00049
00059     GX3Communicator(int priority, const char* serialDevice,
00060                     SerialPort::BaudRate baudRate = SerialPort::BAUD_115200);
00061
00062
00066     void initialize();
00067
00077     virtual void run();
00078
00082     void stop() {mKeepRunning = false;}
00083
00087     void pop() { mQueue.pop();}
00088
00089
00095     bool isEmpty() {return mQueue.isEmpty(); }
00096
00102     unsigned size() {return mQueue.size(); }
00103
00111     packet_ptr &front() { return mQueue.front(); }
00112
00113 private:
00114     GX3Communicator(const GX3Communicator& thread);
00116     GX3Communicator& operator=(const GX3Communicator& rhs);
00118     SerialPort mSerialPort;
00119     SharedQueue<packet_ptr> mQueue;
00120     SerialPort::BaudRate mBaudRate;
00121
00122     volatile bool mKeepRunning;
00123 };
00124
00125 }
00126
00127 #endif // GX3COMMUNICATOR_H

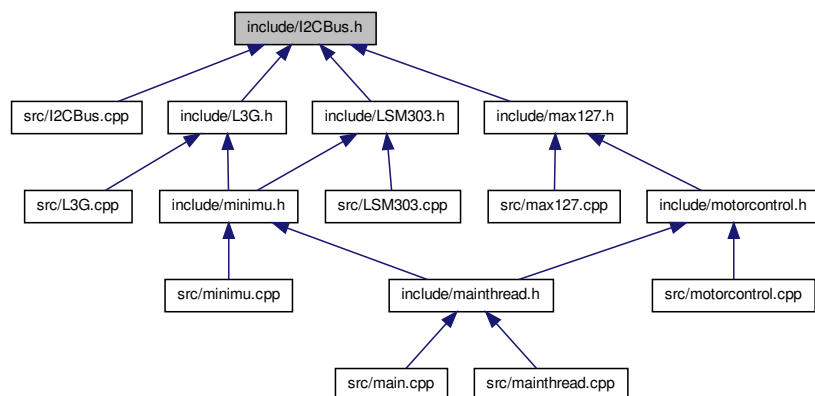
```

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`

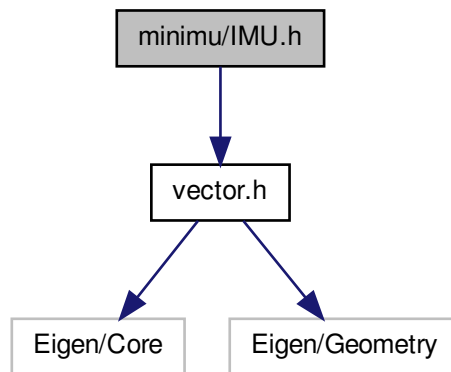
Wrapper class for I2C-bus communication.

9.12 include/I2CBus.h

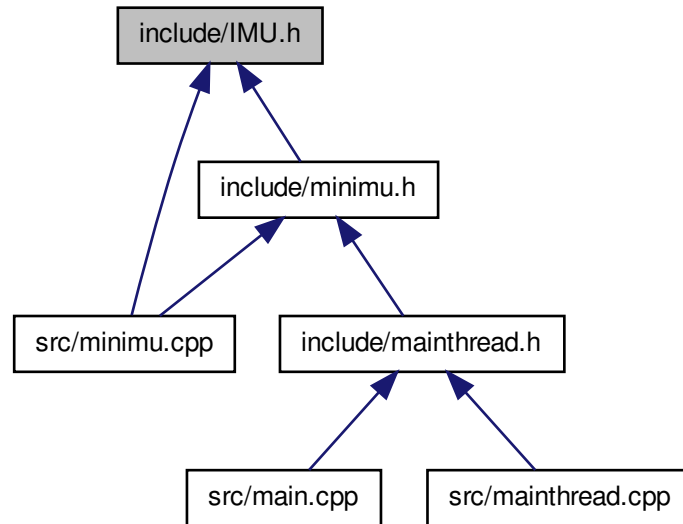
```
00001 #ifndef _I2CBus_h
00002 #define _I2CBus_h
00003
00004 #include <stdint.h>
00005 #include "exceptions.h"
00006
00007
00016 class I2CBus
00017 {
00018 public:
00026     I2CBus(const char * deviceName);
00027
00031     ~I2CBus();
00032
00038     void addressSet(uint8_t address);
00039
00046     void writeByte(uint8_t command, uint8_t data);
00047
00053     void writeByte(uint8_t data);
00054
00061     uint8_t readByte(uint8_t command);
00062
00070     uint8_t readByte();
00071
00078     uint16_t readWord(uint8_t command);
00079
00087     uint16_t readWord();
00088
00098     int tryReadByte(uint8_t command);
00099
00107     void readBlock(uint8_t command, uint8_t size, uint8_t * data);
00108
00109 private:
00110     int fd;
00111 };
00112
00113 #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

```

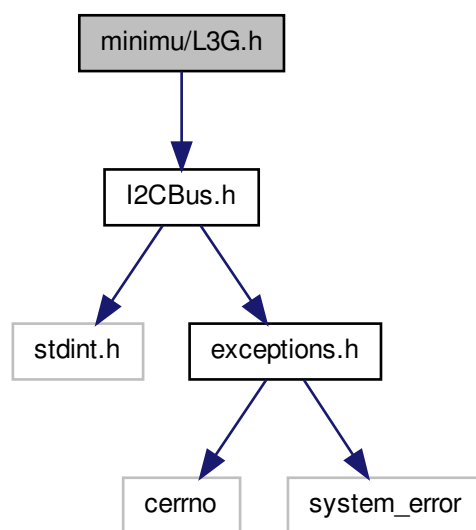
00001 #ifndef __IMU_H
00002 #define __IMU_H
00003
00004 #include "vector.h"
00005
00013 class IMU {
00014 public:
00015     // Scaled readings
00016     virtual vector readMag() = 0; // In body coords, scaled to -1..1 range
00017     virtual vector readAcc() = 0; // In body coords, with units = g
00018     virtual vector readGyro() = 0; // In body coords, with units = rad/sec
00019     void read(){ readAcc(); readMag(); readGyro(); }
00020
00021     virtual void enable() = 0;
00022     // virtual void measureOffsets() = 0;
00023     // virtual void loadCalibration() = 0;
00024
00025     // vector gyro_offset;
00026     // matrix calMatrix;
00027     // vector calOffset;
00028

```

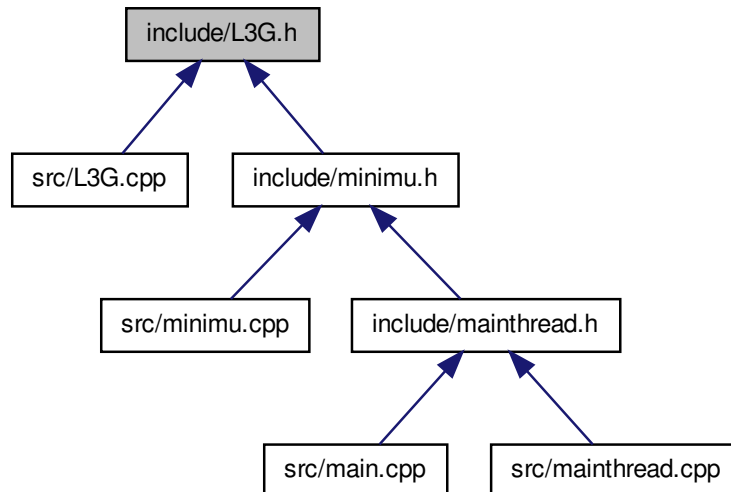
```
00029     int_vector raw_m, raw_a, raw_g;
00030 };
00031
00032 #endif
```

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);

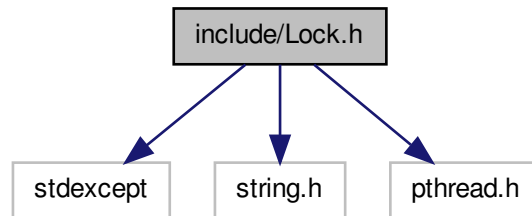
```



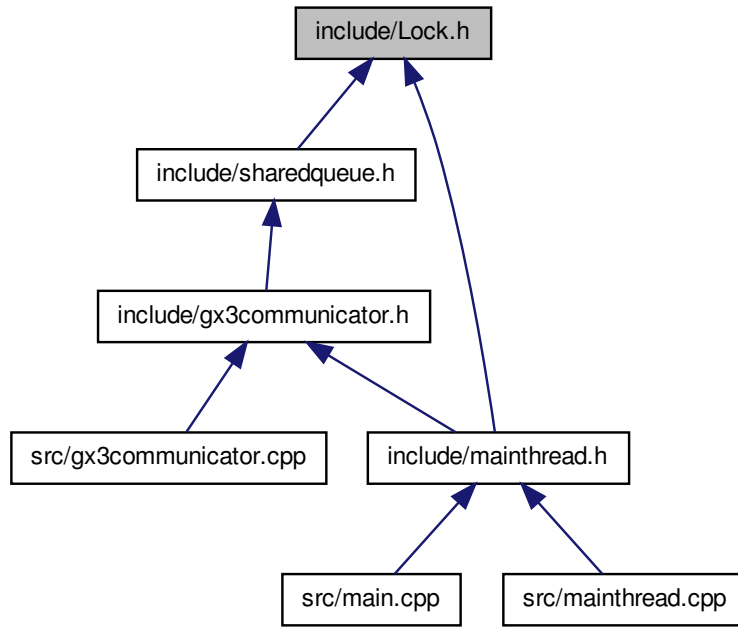
```
00079
00083     void read();
00084
00085 private:
00090     void detectAddress();
00091     I2Cbus i2c;
00092 };
00093
00094 #endif
```

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

```

00001
00011 #ifndef LOCK_H
00012 #define LOCK_H
00013
00014 #include <stdexcept>
00015 #include <string.h>
00016 #include <pthread.h>
00017
00018 namespace USU {
00019
00025 class Lock
00026 {
00027 private:
00028     pthread_mutex_t mMutex;
00030     Lock(const Lock& arg);
00031     Lock& operator=(const Lock& rhs);
00033 public:
00034
00035     Lock();
00037     virtual ~Lock();
00039     void lock();
00041     void unlock();
00042 };
00043
00044 inline
00045 Lock::Lock()
00046 {
00047     int ret;
00048     if ( (ret = pthread_mutex_init(&mMutex, NULL)) != 0)
00049     {
00050         throw std::runtime_error(strerror(errno));
00051     }
00052 }
00053
00054 inline
00055 Lock::~~Lock()
00056 {
00057     int ret;
00058     if ( (ret = pthread_mutex_destroy(&mMutex) ) != 0)
00059     {
00060         throw std::runtime_error(strerror(errno));
00061     }
00062 }
00063
00064
00065 inline
00066 void Lock::lock()
00067 {
00068     pthread_mutex_lock(&mMutex);
00069 }
00070
00071 inline
00072 void Lock::unlock()
00073 {
00074     pthread_mutex_unlock(&mMutex);
00075 }
00076
00077
00092 class ScopedLock
00093 {
00094 private:
00095     Lock &mLock;

```

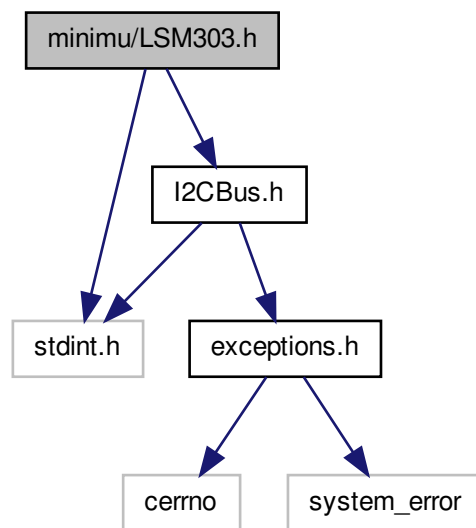
```

00097     ScopedLock(const ScopedLock& thread);
00098     ScopedLock& operator=(const ScopedLock& rhs);
00100 public:
00106     ScopedLock(Lock &lock);
00107
00111     virtual ~ScopedLock();
00112 };
00113
00114 inline
00115 ScopedLock::ScopedLock(Lock &lock)
00116     :mLock(lock)
00117 {
00118     mLock.lock();
00119 }
00120
00121 inline
00122 ScopedLock::~~ScopedLock()
00123 {
00124     mLock.unlock();
00125 }
00126
00127 }
00128
00129 #endif // LOCK_H

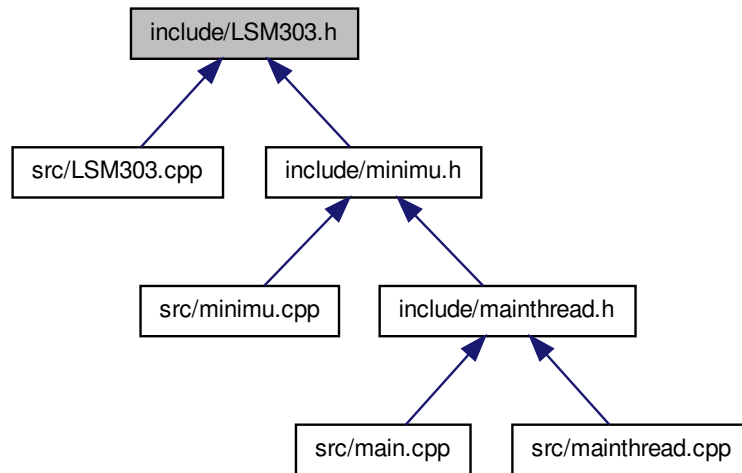
```

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

```

00001 #ifndef LSM303_h
00002 #define LSM303_h
00003
00004 #include <stdint.h>
00005 #include "I2CBus.h"
00006
00007 // register addresses
00008
00009 #define LSM303_CTRL_REG1_A      0x20
00010 #define LSM303_CTRL_REG2_A      0x21
00011 #define LSM303_CTRL_REG3_A      0x22
00012 #define LSM303_CTRL_REG4_A      0x23
00013 #define LSM303_CTRL_REG5_A      0x24
00014 #define LSM303_CTRL_REG6_A      0x25 // DLHC only
00015 #define LSM303_HP_FILTER_RESET_A 0x25 // DLH, DLM only
00016 #define LSM303_REFERENCE_A      0x26
00017 #define LSM303_STATUS_REG_A     0x27
00018
00019 #define LSM303_OUT_X_L_A         0x28
00020 #define LSM303_OUT_X_H_A         0x29
00021 #define LSM303_OUT_Y_L_A         0x2A
00022 #define LSM303_OUT_Y_H_A         0x2B
00023 #define LSM303_OUT_Z_L_A         0x2C
00024 #define LSM303_OUT_Z_H_A         0x2D
00025
00026 #define LSM303_FIFO_CTRL_REG_A   0x2E // DLHC only
00027 #define LSM303_FIFO_SRC_REG_A    0x2F // DLHC only
00028
00029 #define LSM303_INT1_CFG_A         0x30
00030 #define LSM303_INT1_SRC_A         0x31
00031 #define LSM303_INT1_THS_A         0x32
00032 #define LSM303_INT1_DURATION_A   0x33
00033 #define LSM303_INT2_CFG_A         0x34
00034 #define LSM303_INT2_SRC_A         0x35
00035 #define LSM303_INT2_THS_A         0x36
00036 #define LSM303_INT2_DURATION_A   0x37
00037
00038 #define LSM303_CLICK_CFG_A        0x38 // DLHC only
00039 #define LSM303_CLICK_SRC_A        0x39 // DLHC only
00040 #define LSM303_CLICK_THS_A        0x3A // DLHC only
00041 #define LSM303_TIME_LIMIT_A       0x3B // DLHC only
00042 #define LSM303_TIME_LATENCY_A     0x3C // DLHC only
00043 #define LSM303_TIME_WINDOW_A     0x3D // DLHC only
00044
00045 #define LSM303_CRA_REG_M          0x00
00046 #define LSM303_CRB_REG_M          0x01
00047 #define LSM303_MR_REG_M           0x02
00048

```

```

00049 #define LSM303_OUT_X_H_M      0x03
00050 #define LSM303_OUT_X_L_M      0x04
00051 #define LSM303_OUT_Y_H_M      -1    // The addresses of the Y and Z
    magnetometer output registers
00052 #define LSM303_OUT_Y_L_M      -2    // are reversed on the DLM and DLHC
    relative to the DLH.
00053 #define LSM303_OUT_Z_H_M      -3    // These four defines have dummy values
    so the library can
00054 #define LSM303_OUT_Z_L_M      -4    // determine the correct address based on
    the device type.
00055
00056 #define LSM303_SR_REG_M      0x09
00057 #define LSM303_IRA_REG_M      0x0A
00058 #define LSM303_IRB_REG_M      0x0B
00059 #define LSM303_IRC_REG_M      0x0C
00060
00061 #define LSM303_WHO_AM_I_M      0x0F // DLM only
00062
00063 #define LSM303_TEMP_OUT_H_M    0x31 // DLHC only
00064 #define LSM303_TEMP_OUT_L_M    0x32 // DLHC only
00065 #define LSM303DLH_OUT_Y_H_M    0x05
00066 #define LSM303DLH_OUT_Y_L_M    0x06
00067 #define LSM303DLH_OUT_Z_H_M    0x07
00068 #define LSM303DLH_OUT_Z_L_M    0x08
00069
00070 #define LSM303DLM_OUT_Z_H_M    0x05
00071 #define LSM303DLM_OUT_Z_L_M    0x06
00072 #define LSM303DLM_OUT_Y_H_M    0x07
00073 #define LSM303DLM_OUT_Y_L_M    0x08
00074
00075 #define LSM303DLHC_OUT_Z_H_M    0x05
00076 #define LSM303DLHC_OUT_Z_L_M    0x06
00077
00088 class LSM303
00089 {
00090 public:
00091     int a[3];
00092     int m[3];
00101     LSM303(const char * i2cDeviceName);
00102
00107     void enable(void);
00108
00115     void writeAccReg(uint8_t reg, uint8_t value);
00116
00123     uint8_t readAccReg(uint8_t reg);
00124
00131     void writeMagReg(uint8_t reg, uint8_t value);
00132
00139     uint8_t readMagReg(uint8_t reg);
00140
00145     void readAcc(void);
00146
00151     void readMag(void);
00152
00156     void read(void);
00157
00158 private:
00159     I2CBus i2c_mag, i2c_acc;
00168     enum class Device {
00169         LSM303DLH,
00170         LSM303DLM,
00171         LSM303DLHC,
00172     } device;
00173 };
00174
00175 #endif

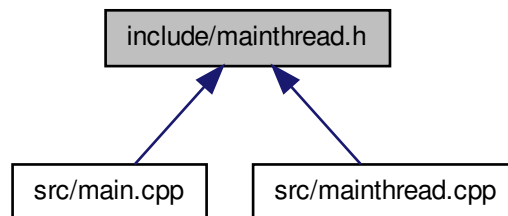
```

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

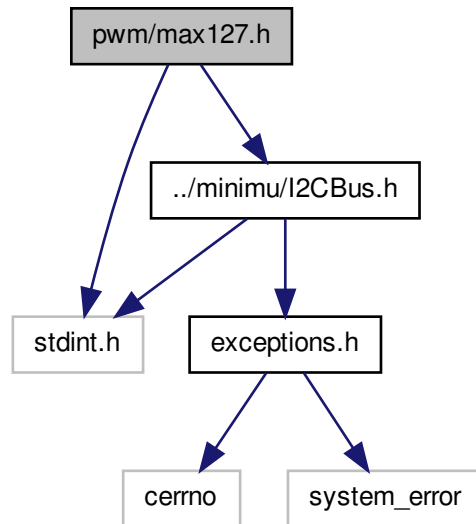
```

00001
00014 #ifndef MAINTHREAD_H
00015 #define MAINTHREAD_H
00016
00017 #include "periodicrtthread.h"
00018 #include "minimu.h"
00019 #include "Lock.h"
00020 #include "gx3communicator.h"
00021 #include "messages.h"
00022 #include "motorcontrol.h"
00023
00024 namespace USU
00025 {
00026
00038 class MainThread : public PeriodicRtThread
00039 {
00040 public:
00041     enum Mode
00042     {
00043         SimpleControl,
00044         CollectPololuData,
00045         CollectMicroStrainData,
00046         CollectData
00047     };
00048
00061     MainThread(int priority, unsigned int period_us, const char* i2cImu, const
char *i2cMotor);
00062
00072     virtual void run();
00073
00078     void stop() { mKeepRunning = false; }
00079
00091     bool getState();
00092
00093     void initializeModeSimpleControl(std::string trajFilename, float pgain);
00094
00095     Mode getMode() const;
00096     void setMode(const Mode &value);
00097
00098 private:
00099     void runSimpleControl();
00100
00101     void runCollectPololu();
00102
00103     void runCollectMicroStrain();
00104
00105     void runCollectBoth();
00106
00107     Mode mMode;
00108
00116     struct Command
00117     {
00118         unsigned int time;
00119         Eigen::Vector3f angVel;
00120     };
00121     std::vector<Command> mCommandList;
00124     MinImu mImu;
00125     GX3Communicator mGX3;
00126     MotorControl mMotors;
00127
00129     bool mState;
00130     Lock mStateLock;
00131     volatile bool mKeepRunning;
00134     MainThread(const MainThread& thread);
00135     MainThread& operator=(const MainThread& rhs);
00136 };
00137
00138 }
00139
00140 #endif // MAINTHREAD_H

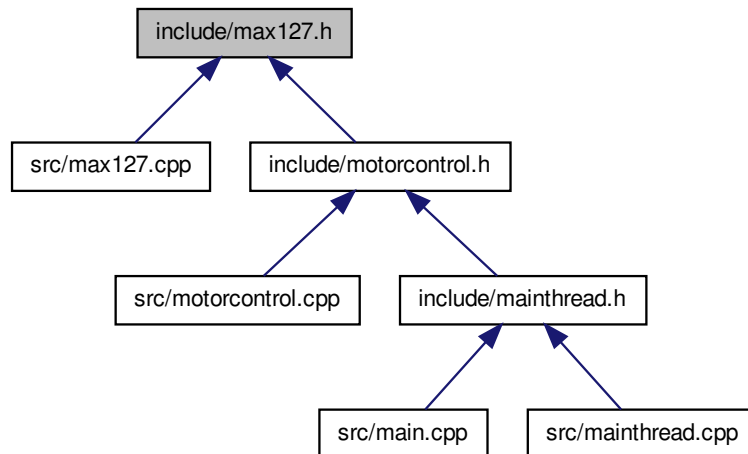
```

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::SELO](#) = 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

```

00001
00011 #ifndef MAX127_H
00012 #define MAX127_H
00013
00014 #include <stdint.h>
00015
00016 #include "I2CBus.h"
00017
00018 namespace USU
00019 {
00020
00027 const uint8_t I2C_ADDRESS = 0b00101000;
00028
00040 const uint8_t CONTROL_BYTE = 0b10000110;
00041 const uint8_t SEL0 = 4;
00052 class Max127
00053 {
00054 public:
00062     Max127(const char *i2cdevice);
00063
00073     int16_t readRaw(uint8_t channel);
00082     float readVoltage(unsigned int channel);
00083
00084 private:
00085     I2CBus mI2c;
00088 };
00089
00090 }
00091
00092 #endif // MAX127_H

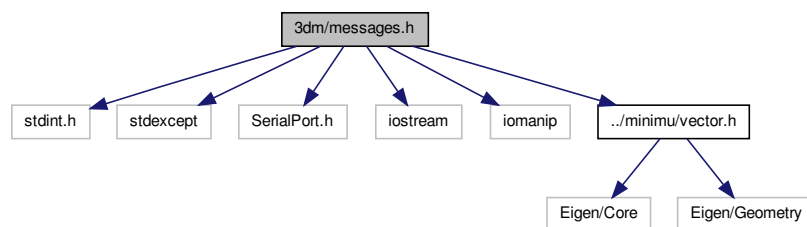
```

9.25 include/messages.h File Reference

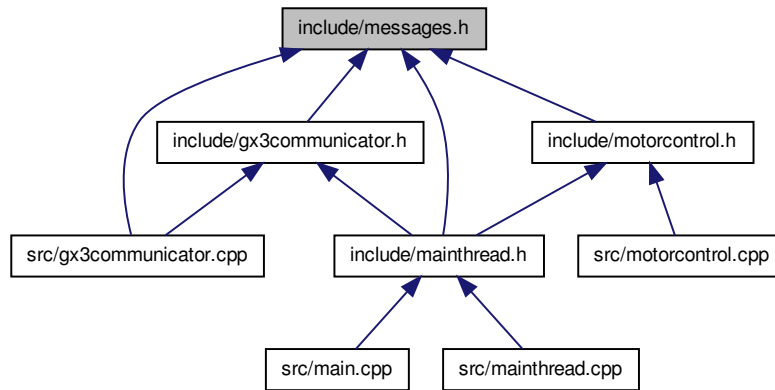
```

#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::SetContinuousMode](#)
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

```

00001
00013 #ifndef MESSAGES_H
00014 #define MESSAGES_H
00015
00016 #include<stdint.h>
00017 #include<stdexcept>
00018 #include<SerialPort.h>
00019 #include<iostream>
00020 #include<iomanip>
00021
00022 #include "vector.h"
00023
00024 namespace USU
00025 {
00026
00027 // Command protocol constants
00028
00029 const uint8_t RAW_ACC_ANG = 0xC1;
00030 const uint8_t ACC_ANG = 0xC2;
00031 const uint8_t DELTA_ANGLE_VEL = 0xC3;
00032 const uint8_t SET_CONTINUOUS_MODE = 0xC4;
00033 const uint8_t ORIENTATION_MATRIX = 0xC5;
00034 const uint8_t ORIENTATION_UPDATE_MAT = 0xC6;
00035 const uint8_t MAG_VEC = 0xC7;
00036 const uint8_t ACC_ANG_ORIENTATION_MAT = 0xC8;
00037 const uint8_t WRITE_ACC_BIAS_CORRECTION = 0xC9;
00038 const uint8_t WRITE_GYRO_BIAS_CORRECTION = 0xCA;
00039 const uint8_t ACC_ANG_MAG_VEC = 0xCB;
00040 const uint8_t ACC_ANG_MAG_VEC_ORIENTATION_MAT = 0xCC;
00041 const uint8_t CAPTURE_GYRO_BIAS = 0xCD;
00042 const uint8_t EULER_ANGLES = 0xCE;
00043 const uint8_t EULER_ANGLES_ANG_RATES = 0xCF;
00044 const uint8_t TRANSFER_TO_NONVOL_MEM = 0xD0;
00045 const uint8_t TEMPERATURES = 0xD1;
00046 const uint8_t GYRO_STABIL_ACC_ANG_MAG = 0xD2;
00047 const uint8_t DELTA_ANGLE_VEL_MAG_VEC = 0xD3;
00048 const uint8_t MODE = 0xD4;
00049 const uint8_t MODE_PRESET = 0xD5;
00050 const uint8_t CONTINUOUS_PRESET = 0xD6;
00051 const uint8_t TIMER = 0xD7;
00052 const uint8_t COMM_SETTINGS = 0xD9;
00053 const uint8_t STATIONARY_TEST = 0xDA;
00054 const uint8_t SAMPLING_SETTINGS = 0xDB;
00055 const uint8_t REALIGN_UP_NORTH = 0xDD;
00056 const uint8_t QUATERNION = 0xDF;
00057 const uint8_t WRITE_WORD_EEPROM = 0xE4;
00058 const uint8_t READ_WORD_EEPROM = 0xE5;
00059 const uint8_t READ_FIRMWARE_VER = 0xE9;
00060 const uint8_t READ_DEVICE_ID = 0xEA;
00061 const uint8_t STOP_CONTINUOUS = 0xFA;
00062 const uint8_t FIRMWARE_UPDATE = 0xFD;
00063 const uint8_t DEVICE_RESET = 0xFE;
00079 class GX3Packet
00080 {
00081 public:
00082     virtual bool readFromSerial(SerialPort &serialPort) = 0;
00089
00098     virtual void print(std::ostream &os) const = 0;
00099
00107     static bool calculateChecksum(uint8_t * buffer, unsigned int length)
00108     {
00109         uint16_t sum = 0;
00110         for(unsigned int i = 0; i<length-2; i++)
00111         {
00112             sum += buffer[i];
00113         }
00114         uint16_t temp = (buffer[length-2] << 8) + buffer[length-1];
00115         return (sum == temp);
00116     }
00117
00118

```

```

00119 protected:
00129     static inline vector createVector(uint8_t * buffer)
00130     {
00131         return vector(*(float*) &buffer[0],
00132                        *(float*) &buffer[4],
00133                        *(float*) &buffer[8]);
00134     }
00135
00142     static inline unsigned int createUInt(uint8_t *buffer)
00143     {
00144         return ( (buffer[0]<<24) + (buffer[1]<<16) + (buffer[2]<<8) + buffer[3]
00145 );
00146     }
00146
00156     static void createMatrix(uint8_t * buffer, matrix& mat)
00157     {
00158         mat << *(float*) &buffer[0], *(float*) &buffer[4], *(float*) &buffer[8],
00159                *(float*) &buffer[12], *(float*) &buffer[16], *(float*) &buffer[20],
00160                *(float*) &buffer[24], *(float*) &buffer[28], *(float*) &buffer[32];
00161     }
00162 };
00163
00173 static std::ostream & operator << (std::ostream & os, const GX3Packet & packet)
00174 {
00175     packet.print(os);
00176     return os;
00177 }
00178
00179
00180
00191 class RawAccAng : public GX3Packet
00192 {
00193 public:
00197     RawAccAng() {}
00198
00199     bool readFromSerial(SerialPort &serialPort)
00200     {
00201         uint8_t buffer[size];
00202         buffer[0] = serialPort.ReadByte();
00203         if(buffer[0] != RAW_ACC_ANG && buffer[0] != ACC_ANG) return false;
00204
00205         serialPort.ReadRaw(&buffer[1], size-1);
00206         if(GX3Packet::calculateChecksum(buffer, size) == false)
00207         {
00208             using namespace std;
00209             return false;
00210         }
00211
00212         acc = createVector(&buffer[1]);
00213         gyro = createVector(&buffer[13]);
00214
00215         timer = createUInt(&buffer[25]);
00216
00217         return true;
00218     }
00219
00228     virtual void print(std::ostream &os) const
00229     {
00230         os << timer/63 << ",\t" << acc(0) << ", " << acc(1) << ", " << acc(2)
00231             << ",\t" << gyro(0) << ", " << gyro(1) << ", " << gyro(2);
00232     }
00233
00234     vector acc;
00235     vector gyro;
00237     unsigned int timer;
00239     enum{size = 31};
00240 };
00241
00252 class AccAngMag : public GX3Packet
00253 {
00254 public:

```

```

00258     AccAngMag() {}
00259
00260     bool readFromSerial(SerialPort &serialPort)
00261     {
00262         unsigned count = 10;
00263         uint8_t buffer[size];
00264         do
00265         {
00266             buffer[0] = serialPort.ReadByte();
00267         } while(--count && buffer[0] != ACC_ANG_MAG_VEC);
00268
00269         if(count == 0)
00270         {
00271             return false; //throw std::runtime_error("Wrong package
00272 identifier");
00273         }
00274
00275         serialPort.ReadRaw(&buffer[1], size-1);
00276         if(GX3Packet::calculateChecksum(buffer, size) == false)
00277         {
00278             using namespace std;
00279             return false;
00280         }
00281
00282         acc = createVector(&buffer[1]);
00283         gyro = createVector(&buffer[13]);
00284         mag = createVector(&buffer[25]);
00285
00286         timer = createUInt(&buffer[37]);
00287
00288         return true;
00289     }
00290
00291     virtual void print(std::ostream &os) const
00292     {
00293         os << timer/63 << ",\t" << acc(0) << ", " << acc(1) << ", " << acc(2)
00294             << ",\t" << mag(0) << ", " << mag(1) << ", " << mag(2)
00295             << ",\t" << gyro(0) << ", " << gyro(1) << ", " << gyro(2);
00296     }
00297
00298     vector acc;
00299     vector gyro;
00300     vector mag;
00301     unsigned int timer;
00302     enum{size = 43};
00303 };
00304
00305 class Quaternion : public GX3Packet
00306 {
00307 public:
00308     Quaternion() {}
00309
00310     bool readFromSerial(SerialPort &serialPort)
00311     {
00312         uint8_t buffer[size];
00313         buffer[0] = serialPort.ReadByte();
00314         if(buffer[0] != QUATERNION)
00315         {
00316             return false; //throw std::runtime_error("Wrong package
00317 identifier");
00318         }
00319
00320         serialPort.ReadRaw(&buffer[1], size-1);
00321         if(GX3Packet::calculateChecksum(buffer, size) == false)
00322         {
00323             return false;
00324         }
00325
00326         quat = quaternion( (float*) &buffer[1]);
00327         timer = createUInt(&buffer[17]);
00328
00329         return true;
00330     }
00331
00332     virtual void print(std::ostream &os) const

```

```

00359     {
00360         os << timer << ",\t" << quat.w() << ", " << quat.x() << ", " << quat.y
    ) << ", " << quat.z();
00361     }
00362
00363     quaternion quat;
00365     unsigned int timer;
00367     enum{size = 23};
00368 };
00369
00370
00379 class AccAngMagOrientationMat : public GX3Packet
00380 {
00381 public:
00385     AccAngMagOrientationMat() {}
00386
00387     bool readFromSerial(SerialPort &serialPort)
00388     {
00389         uint8_t buffer[size];
00390         buffer[0] = serialPort.ReadByte();
00391         if(buffer[0] != ACC_ANG_MAG_VEC_ORIENTATION_MAT) return false;
00392
00393         serialPort.ReadRaw(&buffer[1], size-1);
00394         if(GX3Packet::calculateChecksum(buffer, size) == false) return false;
00395
00396         acc = createVector(&buffer[1]);
00397         gyro = createVector(&buffer[13]);
00398         mag = createVector(&buffer[25]);
00399
00400         createMatrix(&buffer[37], orientation);
00401         timer = createUInt(&buffer[73]);
00402
00403         return true;
00404     }
00405
00406
00415     virtual void print(std::ostream &os) const
00416     {
00417         os << timer << ",\t" << acc(0) << ", " << acc(1) << ", " << acc(2)
00418         << ",\t" << mag(0) << ", " << mag(1) << ", " << mag(2)
00419         << ",\t" << gyro(0) << ", " << gyro(1) << ", " << gyro(2)
00420         << ",\t" << orientation(0,0) << ", " << orientation(0,1) <<
00421         << ",\t" << orientation(1,0) << ", " << orientation(1,1) <<
00422         << ",\t" << orientation(1,2) << ", " << orientation(2,0) << ", " << orientation(2,1) <<
00423         << ",\t" << orientation(2,2);
00424     }
00425
00426     vector acc;
00427     vector gyro;
00428     vector mag;
00430     matrix orientation;
00431     unsigned int timer;
00433     enum {size = 79};
00434 };
00435
00437
00445 class GX3Command
00446 {
00447 public:
00448     virtual bool sendCommand(SerialPort &serialPort) = 0;
00449     virtual bool checkResponse(uint8_t *buffer) = 0;
00450 };
00451
00455 class SetContinuousMode : public GX3Command
00456 {
00457 public:
00467     SetContinuousMode(uint8_t CommandByte = 0)
00468     {
00469         mCommand[0] = SET_CONTINUOUS_MODE;
00470         mCommand[1] = 0xC1;
00471         mCommand[2] = 0x29;

```

```

00472         mCommand[3] = CommandByte;
00473     }
00474
00475     bool sendCommand(SerialPort &serialPort)
00476     {
00477         serialPort.WriteRaw(mCommand, size);
00478         uint8_t buffer[responseSize];
00479         buffer[0] = serialPort.ReadByte();
00480         if(buffer[0] != SET_CONTINUOUS_MODE) return false;
00481
00482         serialPort.ReadRaw(&buffer[1], responseSize-1);
00483         return checkResponse(buffer);
00484     }
00485
00492     bool checkResponse(uint8_t *buffer)
00493     {
00494
00495         if(buffer[0] != SET_CONTINUOUS_MODE) return false;
00496
00497         if(GX3Packet::calculateChecksum(buffer, responseSize) == false)
00498             return false;
00499
00500         if(buffer[1] != mCommand[3]) return false;
00501
00502         return true;
00503     }
00504
00505     enum {size = 4, responseSize = 8 };
00506     uint8_t mCommand[size];
00507 };
00508
00512 class SamplingSettings : public GX3Command
00513 {
00514 public:
00515     enum FunctionSelector{
00516         ReturnOnly=0, Change=1,
00517         ChangeAndSave=2, ChangeWithoutReply=3
00518     };
00519     enum DataConditioning{
00520         FlagCalcOrientation      = 0x01,
00521         FlagEnableConingSculling = 0x02,
00522         FlagDefault              = 0x03,
00523         FlagFloatLittleEndian    = 0x10,
00524         FlagSuppressNaN          = 0x20,
00525         FlagFiniteSizeCorrection = 0x40,
00526         FlagDisableMag           = 0x100,
00527         FlagDisableMagNorthComp  = 0x400,
00528         FlagDisableGravComp      = 0x800,
00529         FlagEnableQuaternion     = 0x1000
00530     };
00531 };
00532
00533     SamplingSettings(FunctionSelector funSel, uint16_t samplingPeriod_ms = 10,
00534                     uint16_t dataCondFlags = SamplingSettings::FlagDefault,
00535                     uint8_t gyroAccFilter = 15, uint8_t magFilter = 17,
00536                     uint16_t upCompensation = 10, uint16_t northCompensation =
00537     10,
00538                     uint8_t magPower = 0)
00539     {
00540         mCommand[0] = SAMPLING_SETTINGS;
00541         mCommand[1] = 0xA8;
00542         mCommand[2] = 0xB9;
00543         mCommand[3] = (uint8_t) funSel;
00544         mCommand[4] = (samplingPeriod_ms >> 8);
00545         mCommand[5] = (samplingPeriod_ms & 0x00FF);
00546         mCommand[6] = (dataCondFlags >> 8);
00547         mCommand[7] = (dataCondFlags & 0x00FF);
00548         mCommand[8] = gyroAccFilter;
00549         mCommand[9] = magFilter;
00550         mCommand[10] = (upCompensation >> 8);
00551         mCommand[11] = (upCompensation & 0x00FF);
00552         mCommand[12] = (northCompensation >> 8);
00553         mCommand[13] = (northCompensation & 0x00FF);

```



```

00587         mCommand[14]= magPower;
00588         mCommand[15]= 0;
00589         mCommand[16]= 0;
00590         mCommand[17]= 0;
00591         mCommand[18]= 0;
00592         mCommand[19]= 0;
00593     }
00594 }
00595
00596 bool sendCommand(SerialPort &serialPort)
00597 {
00598     serialPort.WriteRaw(mCommand, size);
00599     uint8_t buffer[responseSize];
00600     buffer[0] = serialPort.ReadByte();
00601     if(buffer[0] != SAMPLING_SETTINGS) return false;
00602
00603     serialPort.ReadRaw(&buffer[1], responseSize-1);
00604     return checkResponse(buffer);
00605 }
00606
00613 bool checkResponse(uint8_t *buffer)
00614 {
00615     if(GX3Packet::calculateChecksum(buffer, responseSize) == false)
00616         return false;
00617
00618     if(buffer[0] != SAMPLING_SETTINGS) return false;
00619
00620     for(int i=1; i<11; i++)
00621     {
00622         if(buffer[i] != mCommand[i+3]) return false;
00623     }
00624
00625     return true;
00626 }
00627
00628 enum {size = 20, responseSize = 19};
00629 uint8_t mCommand[size];
00631 };
00632
00635 }
00636 #endif // MESSAGES_H

```

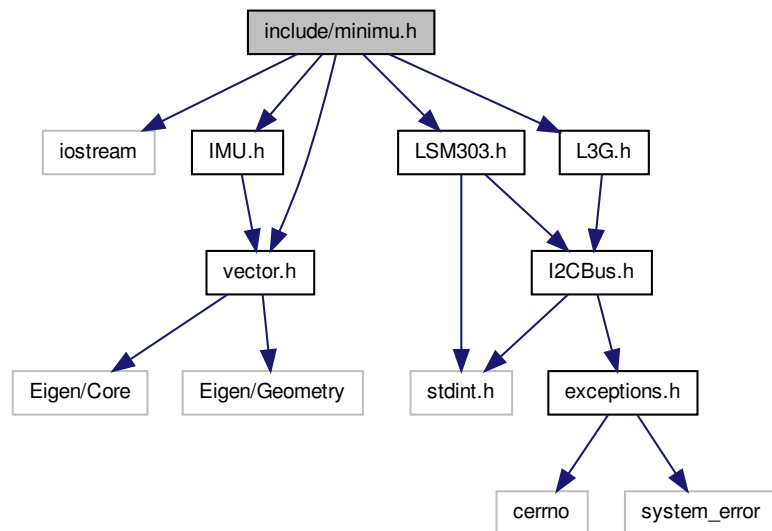
9.27 include/minimu.h File Reference

```

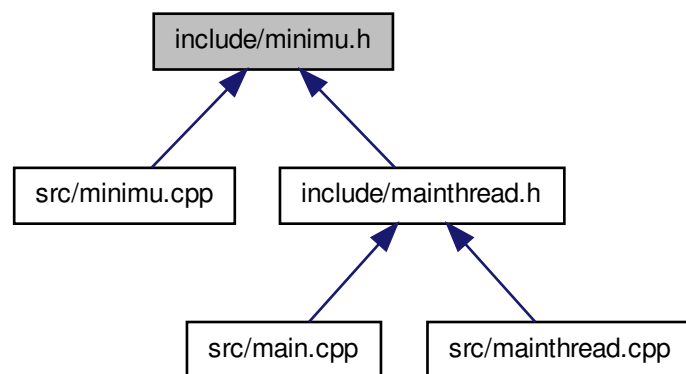
#include <iostream> #include "IMU.h" #include "LSM303.h" ×
#include "L3G.h" #include "vector.h" Include dependency graph for

```

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

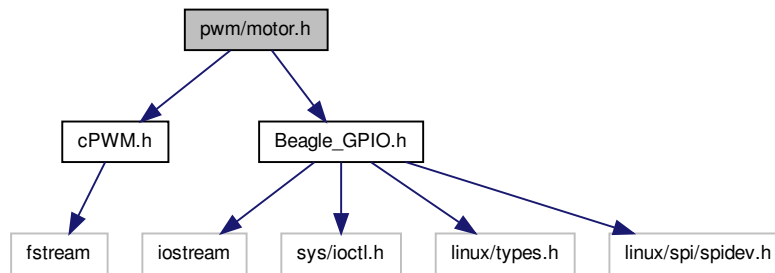
```

00001
00011 #ifndef MINIMU_H
00012 #define MINIMU_H
00013
00014 #include <iostream>
00015
00016 #include "IMU.h"
00017 #include "LSM303.h"
00018 #include "L3G.h"
00019 #include "vector.h"
00020
00021 namespace USU
00022 {
00023
00024
00025
00032 class MinImu : public IMU
00033 {
00034 public:
00035     LSM303 compass;
00036     L3G gyro;
00045     MinImu(const char * i2cDeviceName);
00046
00054     virtual vector readMag(); // In body coords, currently without scaling
00055
00061     virtual vector readAcc(); // In body coords, with units = g
00062
00068     virtual vector readGyro(); // In body coords, with units = degrees/sec
00069
00073     virtual void enable();
00074 };
00075
00076 }
00077
00078 #endif // MINIMU_H

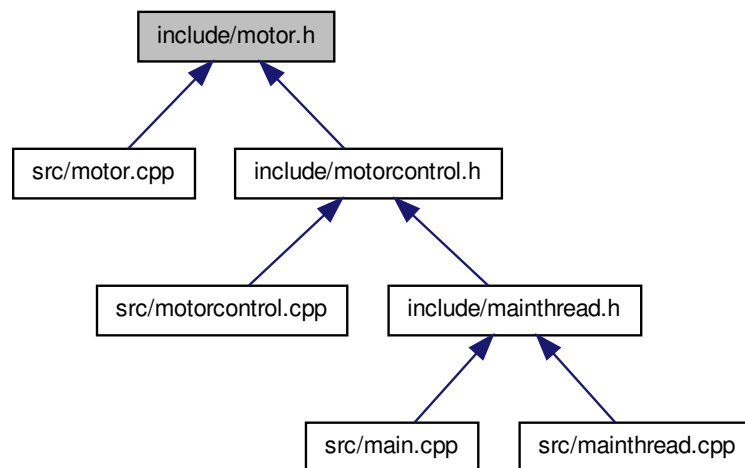
```

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

```

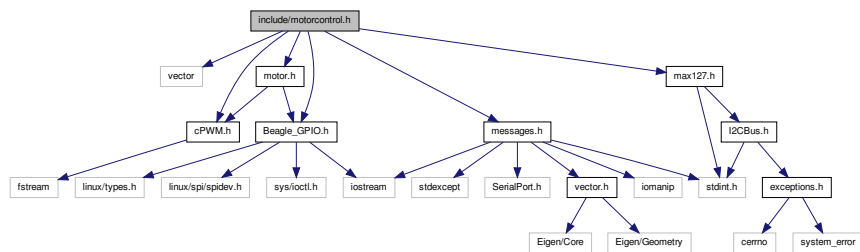
```

00065 private:
00066     Beagle_GPIO& mBeagleGpio;
00067     Beagle_GPIO::Pins mClockwise;
00068     Beagle_GPIO::Pins mCounterClockwise;
00070     cPWM &mPwm;
00071     SetDutyCycle mSetDutyCycle;
00072     int mSpeed;
00073 };
00074
00075 }
00076
00077 #endif // MOTOR_H

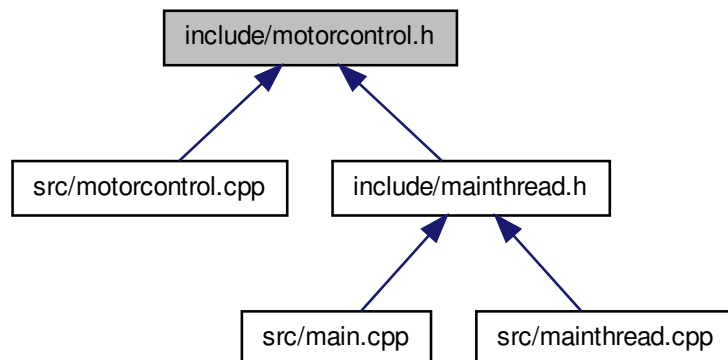
```

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

```

00001
00012 #ifndef MOTORCONTROL_H
00013 #define MOTORCONTROL_H
00014
00015 #include <vector>
00016
00017 #include "cPWM.h"
00018 #include "Beagle_GPIO.h"
00019 #include "motor.h"
00020 #include "max127.h"
00021 #include "messages.h"
00022
00023 namespace USU
00024 {
00025
00026     00038 class MotorControl
00027     {
00028     public:
00029
00030         00041
00031         00042
00032         00051 MotorControl(const char* i2cDevice="/dev/i2c-3");
00033         00052
00034         00053 virtual ~MotorControl();
00035         00054
00036         00062 void calculateControlResponse(Quaternion state);
00037         00063
00038         00069 void controlFromGyro(const Eigen::Vector3f &gyro);
00039         00070
00040         00077 void setMotor(int motor, int dutyCycle);
00041         00078
00042         00079
00043         00087 void getAnalog(int motor, float &aOut1, float &aOut2);
00044         00088
00045         00095 void getAnalogS(float * aOut1, float* aOut2);
00046         00096
00047         00102 void getDutyCycles(int* dc);

```

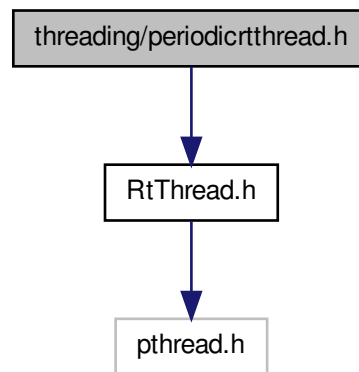
```

00103
00104     float getPGain() const;
00105     void setPGain(float value);
00106
00107     Eigen::Vector3f getSetValue() const;
00108     void setSetValue(const Eigen::Vector3f value);
00109
00110 private:
00111
00112     cPWM mPwm1;
00113     cPWM mPwm2;
00115     Beagle_GPIO mBeagleGpio;
00116     Motor *mMotor[4];
00118     Max127 mAnalog;
00119     float mPGain;
00120     Eigen::Vector3f mSetValue;
00121
00122     MotorControl(const MotorControl& thread);
00124     MotorControl& operator=(const MotorControl& rhs);
00125 };
00126
00127 }
00128
00129 #endif // MOTORCONTROL_H

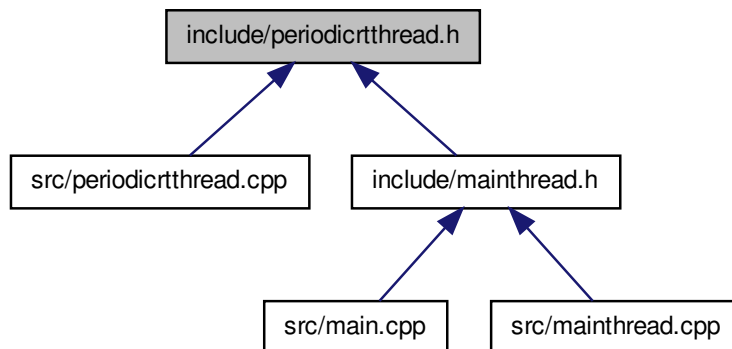
```

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

```

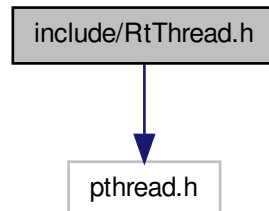
00001
00012 #ifndef PERIODICRTTHREAD_H
00013 #define PERIODICRTTHREAD_H

```

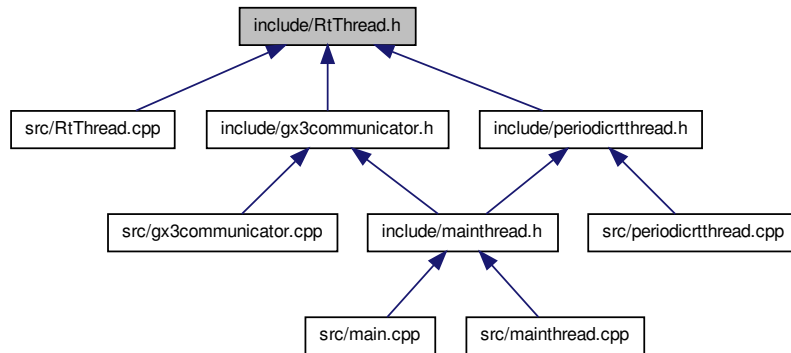
```
00014
00015 #include "RtThread.h"
00016
00017 namespace USU {
00018
00020
00032 class PeriodicRtThread : public RtThread
00033 {
00034 private:
00035     int mTimerFd;
00045     unsigned int mMissedWakeUps;
00046     unsigned int mPeriod_us;
00048     PeriodicRtThread(const PeriodicRtThread& thread);
00049     PeriodicRtThread& operator=(const PeriodicRtThread& rhs);
00051 protected:
00057     void makeThreadPeriodic();
00058
00066     void waitPeriod();
00067
00068 public:
00078     PeriodicRtThread(int priority = 0, unsigned int period_us = 1000000);
00079
00086     virtual void run() = 0;
00087 };
00088
00089 }
00090
00091 #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

```

00001
00012 #ifndef RTTHREAD_H
00013 #define RTTHREAD_H
00014
00015 #include<pthread.h>
00016
00018

```

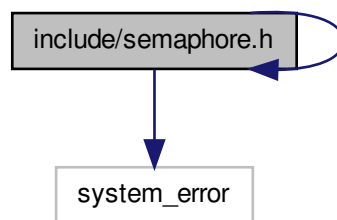
```

00019 namespace USU {
00020
00032 class RtThread
00033 {
00034
00035 private:
00036     pthread_attr_t mAttr;
00037     int mPriority;
00039     RtThread(const RtThread& thread);
00040     RtThread& operator=(const RtThread& rhs);
00042 protected:
00043     pthread_t mId;
00044     bool mStarted;
00045     void *mArgs;
00054     static void *exec(void * thr);
00055
00056 public:
00065     RtThread(int priority = 0);
00066
00074     virtual ~RtThread();
00075
00081     pthread_t getThreadId() const;
00082
00088     int getPriority() const;
00089
00097     void start(void * args = NULL);
00098
00105     bool join(int timeout_ms = 0);
00106
00113     virtual void run() = 0;
00114 };
00115
00116 }
00117
00118 #endif // RTTHREAD_H

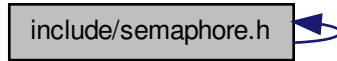
```

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

```

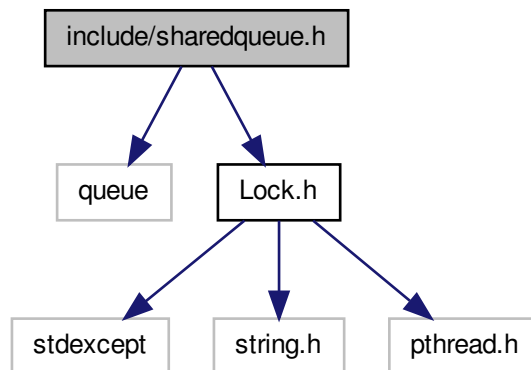
00001
00011 #ifndef SEMAPHORE_H
00012 #define SEMAPHORE_H
00013
00014 #include <semaphore.h>
00015 #include <system_error>
00016
00017 namespace USU
00018 {
00019
00020
00027 class Semaphore
00028 {
00029     private:
00030
00031         sem_t mSem;
00033         Semaphore(const Semaphore& arg);
00034         Semaphore& operator=(const Semaphore& rhs);

```

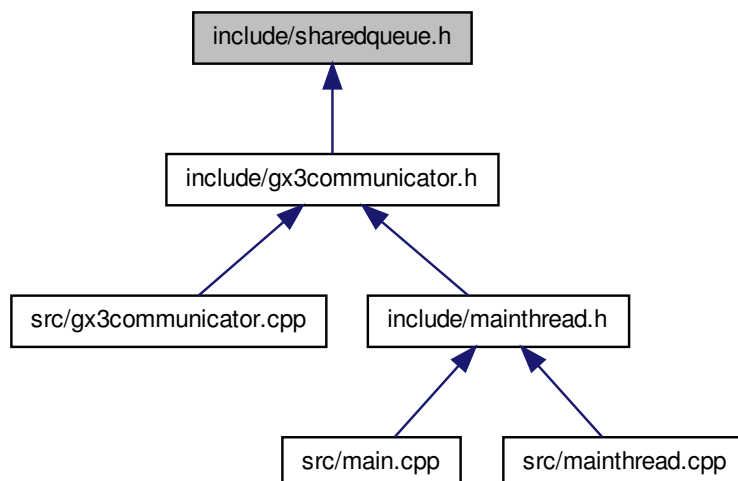
```
00036 public:
00037
00038     Semaphore();
00040     virtual ~Semaphore();
00042     void post();
00051     void wait();
00052
00060     bool tryWait();
00061 };
00062
00063 Semaphore::Semaphore()
00064 {
00065     int ret = sem_init(&mSem, 0, 0);
00066     if(ret != 0)
00067     {
00068         throw std::system_error(ret, std::system_category());
00069     }
00070 }
00071
00072 Semaphore::~Semaphore()
00073 {
00074     int ret = sem_close(&mSem);
00075     if(ret != 0)
00076     {
00077         throw std::system_error(ret, std::system_category());
00078     }
00079 }
00080
00081 inline
00082 void Semaphore::post()
00083 {
00084     sem_post(&mSem);
00085 }
00086
00087 inline
00088 void Semaphore::wait()
00089 {
00090     sem_wait(&mSem);
00091 }
00092
00093 inline
00094 bool Semaphore::tryWait()
00095 {
00096     return (!sem_trywait(&mSem) ? true : false);
00097 }
00098
00099 }
00100
00101 #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

```

00001
00012 #ifndef SHAREDQUEUE_H
00013 #define SHAREDQUEUE_H
00014
00015 #include <queue>
00016 using std::queue;
00017
00018 #include "Lock.h"
00019
00020 namespace USU
00021 {
00034 template <class T>
00035 class SharedQueue
00036 {
00037 public:
00042 // SharedQueue();
00043
00051 void push(const T& newElement)
00052 {
00053     ScopedLock scLock(mLock);
00054     mQueue.push(newElement);
00055 }
00056
00064 void pop()
00065 {
00066     ScopedLock scLock(mLock);
00067     mQueue.pop();
00068 }
00069
00077 T& front()
00078 {
00079     ScopedLock scLock(mLock);
00080     return mQueue.front();
00081 }
00082
00088 bool isEmpty()

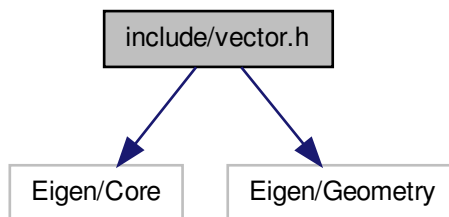
```



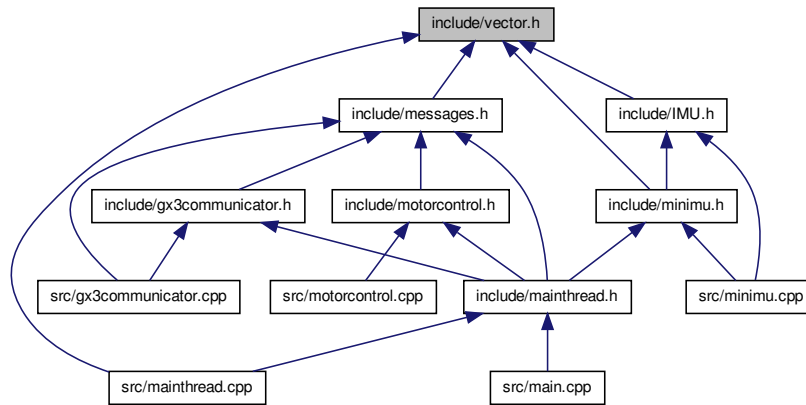
```
00089     {  
00090         ScopedLock scLock(mLock);  
00091         return mQueue.empty();  
00092     }  
00093  
00094     int size()  
00095     {  
00096         ScopedLock scLock(mLock);  
00097         return mQueue.size();  
00098     }  
00099  
00100 private:  
00101     Lock mLock;  
00102     queue<T> mQueue;  
00103 };  
00104  
00105 }  
00106  
00107 #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

```
00001 #ifndef _vector_h
```

```

00002 #define _vector_h
00003
00004 #include "Eigen/Core"
00005 #include "Eigen/Geometry"
00006 typedef Eigen::Vector3f vector;
00007 typedef Eigen::Vector3i int_vector;
00008 typedef Eigen::Matrix3f matrix;
00009 typedef Eigen::Quaternionf quaternion;
00010
00011 static inline vector vector_from_ints(int (*ints)[3])
00012 {
00013     return vector((float)(*ints)[0], (float)(*ints)[1], (float)(*ints)[2]);
00014 }
00015
00016 static inline int_vector int_vector_from_ints(int (*ints)[3])
00017 {
00018     return int_vector((*ints)[0], (*ints)[1], (*ints)[2]);
00019 }
00020
00021 #endif

```

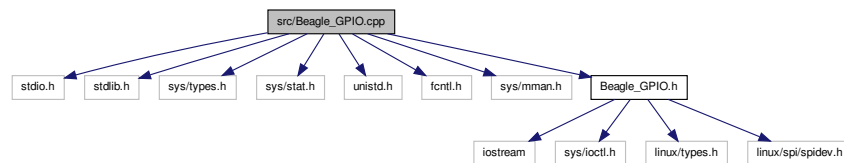
9.43 src/Beagle_GPIO.cpp File Reference

```

#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

```

00001 /*****
00002 ** Beagle Bone GPIO Library **
00003 **
00004 **      Francois Sugny      **
00005 **      01/07/12           **
00006 **
00007 **      v1.0               **
00008 *****/
00009
00010 //=====
00011 //=====
00012
00013 #include <stdio.h>
00014 #include <stdlib.h>
00015 #include <sys/types.h>
00016 #include <sys/stat.h>
00017 #include <unistd.h>
00018 #include <fcntl.h>
00019 #include <sys/mman.h>
00020
00021 //=====

```

```

00022 //=====
00023
00024 #include "Beagle_GPIO.h"
00025
00026 //=====
00027 //=====
00028
00029 const int Beagle_GPIO::GPIO_Pin_Bank[] =
00030 {
00031     -1, -1, 1, 1, 1, // P8_1 -> P8_5
00032     1, 2, 2, 2, 2, // P8_6 -> P8_10
00033     1, 1, 0, 0, 1, // P8_11 -> P8_15
00034     1, 0, 2, 0, 1, // P8_16 -> P8_20
00035     1, 1, 1, 1, 1, // P8_21 -> P8_25
00036     1, 2, 2, 2, 2, // P8_26 -> P8_30
00037     0, 0, 0, 2, 0, // P8_31 -> P9_35
00038     2, 2, 2, 2, 2, // P8_36 -> P8_40
00039     2, 2, 2, 2, 2, // P8_41 -> P8_45
00040     2, // P8_46
00041     -1, -1, -1, -1, -1, // P9_1 -> P9_5
00042     -1, -1, -1, -1, -1, // P9_6 -> P9_10
00043     0, 1, 0, 1, 1, // P9_11 -> P9_15
00044     1, 0, 0, 0, 0, // P9_16 -> P9_20
00045     0, 0, 1, 0, 3, // P9_21 -> P9_25
00046     0, 3, 3, 3, 3, // P9_26 -> P9_30
00047     3, -1, -1, -1, -1, // P9_31 -> P9_35
00048     -1, -1, -1, -1, -1, // P9_36 -> P9_40
00049     0, 0, -1, -1, -1, // P9_41 -> P9_45
00050     -1 // P9_46
00051 };
00052
00053 //=====
00054 //=====
00055
00056 const int Beagle_GPIO::GPIO_Pin_Id[] =
00057 {
00058     -1, -1, 6, 7, 2, // P8_1 -> P8_5
00059     3, 2, 3, 5, 4, // P8_6 -> P8_10
00060     13, 12, 23, 26, 15, // P8_11 -> P8_15
00061     14, 27, 1, 22, 31, // P8_16 -> P8_20
00062     30, 5, 4, 1, 0, // P8_21 -> P8_25
00063     29, 22, 24, 23, 25, // P8_26 -> P8_30
00064     10, 11, 9, 17, 8, // P8_31 -> P9_35
00065     16, 14, 15, 12, 13, // P8_36 -> P8_40
00066     10, 11, 8, 9, 6, // P8_41 -> P8_45
00067     7, // P8_46
00068     -1, -1, -1, -1, -1, // P9_1 -> P9_5
00069     -1, -1, -1, -1, -1, // P9_6 -> P9_10
00070     30, 28, 31, 18, 16, // P9_11 -> P9_15
00071     19, 5, 4, 13, 12, // P9_16 -> P9_20
00072     3, 2, 17, 15, 21, // P9_21 -> P9_25
00073     14, 19, 17, 15, 16, // P9_26 -> P9_30
00074     14, -1, -1, -1, -1, // P9_31 -> P9_35
00075     -1, -1, -1, -1, -1, // P9_36 -> P9_40
00076     20, 7, -1, -1, -1, // P9_41 -> P9_45
00077     -1 // P9_46
00078 };
00079
00080 //=====
00081 //=====
00082
00083 // Pad Control Register
00084 const unsigned long Beagle_GPIO::GPIO_Pad_Control[] =
00085 {
00086     0x0000, 0x0000, 0x0818, 0x081C, 0x0808, // P8_1 -> P8_5
00087     0x080C, 0x0890, 0x0894, 0x089C, 0x0898, // P8_6 -> P8_10
00088     0x0834, 0x0830, 0x0824, 0x0828, 0x083C, // P8_11 -> P8_15
00089     0x0838, 0x082C, 0x088C, 0x0820, 0x0884, // P8_16 -> P8_20
00090     0x0880, 0x0814, 0x0810, 0x0804, 0x0800, // P8_21 -> P8_25
00091     0x087C, 0x08E0, 0x08E8, 0x08E4, 0x08EC, // P8_26 -> P8_30
00092     0x08D8, 0x08DC, 0x08D4, 0x08CC, 0x08D0, // P8_31 -> P8_35
00093     0x08C8, 0x08C0, 0x08C4, 0x08B8, 0x08BC, // P8_36 -> P8_40
00094     0x08B0, 0x08B4, 0x08A8, 0x08AC, 0x08A0, // P8_41 -> P8_45
00095     0x08A4,

```

```

00096 // P8_46
00097 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // P9_1 -> P9_5
00098 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // P9_6 -> P9_10
00099 0x0870, 0x0878, 0x0874, 0x0848, 0x0840, // P9_11 -> P9_15
00100 0x084C, 0x095C, 0x0958, 0x097C, 0x0978, // P9_16 -> P9_20
00101 0x0954, 0x0950, 0x0844, 0x0984, 0x09AC, // P9_21 -> P9_25
00102 0x0980, 0x09A4, 0x099C, 0x0994, 0x0998, // P9_26 -> P9_30
00103 0x0990, 0x0000, 0x0000, 0x0000, 0x0000, // P9_31 -> P9_35
00104 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, // P9_36 -> P9_40
00105 0x09B4, 0x0964, 0x0000, 0x0000, 0x0000, // P9_41 -> P9_45
00106 0x0000
00107 // P9_46
00108 };
00109 //=====
00110 //=====
00111 const unsigned long Beagle_GPIO::GPIO_Control_Module_Registers = 0x44E10000;
00112
00113 //=====
00114 //=====
00115
00116 const unsigned long Beagle_GPIO::GPIO_Base[] =
00117 {
00118     0x44E07000, // GPIO0
00119     0x4804C000, // GPIO1
00120     0x481AC000, // GPIO2
00121     0x481AE000 // GPIO3
00122 };
00123
00124 //=====
00125 //=====
00126
00127 Beagle_GPIO::Beagle_GPIO()
00128 {
00129     GPIO_PRINT( "Beagle_GPIO::Beagle_GPIO()" );
00130
00131     // Not initialized by default
00132     m_active = false;
00133
00134     // Opening /dev/mem first
00135     GPIO_PRINT( "Opening /dev/mem" );
00136     m_gpio_fd = open( "/dev/mem", O_RDWR | O_SYNC );
00137     if ( m_gpio_fd < 0 )
00138     {
00139         GPIO_ERROR( "Cannot open /dev/mem" );
00140         return;
00141     }
00142
00143     // Map Control Module
00144     m_controlModule = (unsigned long *)mmap( NULL, 0x1FFF, PROT_READ
| PROT_WRITE, MAP_SHARED, m_gpio_fd, GPIO_Control_Module_Registers );
00145     if ( m_controlModule == MAP_FAILED )
00146     {
00147         GPIO_ERROR( "Control Module Mapping failed" );
00148         return;
00149     }
00150
00151     // Now mapping the GPIO registers
00152     for ( int i=0; i<4; ++i)
00153     {
00154         // Map a GPIO bank
00155         m_gpio[i] = (unsigned long *)mmap( NULL, 0xFFF,
PROT_READ | PROT_WRITE, MAP_SHARED, m_gpio_fd, GPIO_Base[i] );
00156         if ( m_gpio[i] == MAP_FAILED )
00157         {
00158             GPIO_ERROR( "GPIO Mapping failed
for GPIO Module " << i );
00159             return;
00160         }
00161     }
00162
00163     // Init complete and successfull
00164     m_active = true;

```

```

00165
00166             GPIO_PRINT( "Beagle GPIO Initialized" );
00167     }
00168
00169     //=====
00170     //=====
00171
00172     Beagle_GPIO::~Beagle_GPIO()
00173     {
00174         //GPIO_PRINT( "BeAGLe_GPIO::~~Beagle_GPIO()" );
00175         if ( m_active && m_gpio_fd)
00176             close( m_gpio_fd );
00177     }
00178
00179     //=====
00180     //=====
00181
00182     // Configure pin as input/output
00183     Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::configurePin( unsigned short _pin,
00184         Beagle_GPIO_Direction _direction )
00185     {
00186         if ( !m_active )
00187             return kFail;
00188
00189         gp_assert( GPIO_Pin_Bank[_pin]>=0);
00190         gp_assert( GPIO_Pin_Id[_pin]>=0);
00191
00192         // Set Pin as GPIO on the pad control
00193         m_controlModule[GPIO_Pad_Control[_pin]/4] |= 0x07;
00194
00195         unsigned long v = 0x1 << GPIO_Pin_Id[_pin];
00196
00197         if ( _direction == kINPUT)
00198         {
00199             m_gpio[GPIO_Pin_Bank[_pin]][kOE/4] |= v;
00200         }
00201         else
00202         {
00203             m_gpio[GPIO_Pin_Bank[_pin]][kOE/4] &= ~v;
00204         }
00205
00206         // Disable Interrupts by default
00207         m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_0/4] |= v;
00208         m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_1/4] |= v;
00209
00210         return kSuccess;
00211     }
00212     //=====
00213     //=====
00214
00215     // Enable/Disable interrupts for the pin
00216     Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::enablePinInterrupts( unsigned
00217         short _pin, bool _enable )
00218     {
00219         if ( !m_active )
00220             return kFail;
00221
00222         gp_assert( GPIO_Pin_Bank[_pin]>=0);
00223         gp_assert( GPIO_Pin_Id[_pin]>=0);
00224
00225         // Set Pin as GPIO on the pad control
00226         m_controlModule[GPIO_Pad_Control[_pin]/4] |= 0x07;
00227
00228         unsigned long v = 0x1 << GPIO_Pin_Id[_pin];
00229
00230         if ( _enable )
00231         {
00232             m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_SET_0/4] |
00233             = v;
00234             m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_SET_1/4] |
00235             = v;
00236         }
00237         else

```

```

00235         {
00236             m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_0/4] |
= v;
00237             m_gpio[GPIO_Pin_Bank[_pin]][kIRQSTATUS_CLR_1/4] |
= v;
00238         }
00239
00240         return kSuccess;
00241
00242     }
00243
00244     //=====
00245     //=====
00246
00247     // Write a value to a pin
00248     Beagle_GPIO::Beagle_GPIO_Status Beagle_GPIO::writePin( unsigned short _pin,
unsigned char _value )
00249     {
00250         gp_assert( GPIO_Pin_Bank[_pin]>=0);
00251         gp_assert( GPIO_Pin_Id[_pin]>=0);
00252
00253         unsigned long v = (_value & 0x01) << GPIO_Pin_Id[_pin];
00254         unsigned long mask = 0x1 << GPIO_Pin_Id[_pin];
00255
00256         // Remove bit
00257         m_gpio[GPIO_Pin_Bank[_pin]][kDATAOUT/4] &= ~mask;
00258         // Assign new bit value
00259         m_gpio[GPIO_Pin_Bank[_pin]][kDATAOUT/4] |= v;
00260
00261         return kSuccess;
00262     }
00263
00264     //=====
00265     //=====
00266
00267     // Read a value from a pin
00268     unsigned char Beagle_GPIO::readPin( unsigned short _pin )
00269     {
00270         gp_assert( GPIO_Pin_Bank[_pin]>=0);
00271         gp_assert( GPIO_Pin_Id[_pin]>=0);
00272
00273         unsigned long bit = GPIO_Pin_Id[_pin];
00274         return (m_gpio[GPIO_Pin_Bank[_pin]][kDATAIN/4] & (0x1 << bit)) >
> bit;
00275     }
00276
00277     //=====
00278     //=====
00279
00280     // Default SPI Device for the beaglebone
00281     static const char *spi_device = "/dev/spidev2.0";
00282
00283     // Open SPI Channel
00284     void Beagle_GPIO::openSPI( unsigned char _mode,
                                unsigned char _bits,
                                unsigned long _speed,
                                unsigned short _delay )
00285     {
00286
00287         GPIO_PRINT( "Opening SPI Device" );
00288         m_spi_fd = open( spi_device, O_RDWR );
00289         if ( m_spi_fd < 0 )
00290         {
00291             GPIO_ERROR( "Error opening SPI Device" );
00292             return;
00293         }
00294
00295         int ret = 0;
00296
00297         // Save settings
00298         m_spi_mode = _mode;
00299         m_spi_bits = _bits;
00300         m_spi_speed = _speed;
00301         m_spi_delay = _delay;
00302
00303
00304

```

```

00305         m_spi_buffer_rx = new unsigned char[65536];
00306
00307         // SPI Mode
00308         ret = ioctl(m_spi_fd, SPI_IOC_WR_MODE, &m_spi_mode);
00309         if (ret == -1)
00310         {
00311             GPIO_ERROR( "Error setting SPI Mode");
00312             return;
00313         }
00314
00315         ret = ioctl(m_spi_fd, SPI_IOC_RD_MODE, &m_spi_mode);
00316         if (ret == -1)
00317         {
00318             GPIO_ERROR( "Error getting SPI Mode");
00319             return;
00320         }
00321
00322         // SPI Bits Per Word
00323         ret = ioctl(m_spi_fd, SPI_IOC_WR_BITS_PER_WORD, &m_spi_bits);
00324         if (ret == -1)
00325         {
00326             GPIO_ERROR( "Error setting SPI Bits Per Word");
00327             return;
00328         }
00329
00330         ret = ioctl(m_spi_fd, SPI_IOC_RD_BITS_PER_WORD, &m_spi_bits);
00331         if (ret == -1)
00332         {
00333             GPIO_ERROR( "Error getting SPI Bits Per Word");
00334             return;
00335         }
00336
00337         // SPI Max Speed
00338         ret = ioctl(m_spi_fd, SPI_IOC_WR_MAX_SPEED_HZ, &m_spi_speed);
00339         if (ret == -1)
00340         {
00341             GPIO_ERROR( "Error setting SPI Max Speed");
00342             return;
00343         }
00344
00345         ret = ioctl(m_spi_fd, SPI_IOC_RD_MAX_SPEED_HZ, &m_spi_speed);
00346         if (ret == -1)
00347         {
00348             GPIO_ERROR( "Error getting SPI Max Speed");
00349             return;
00350         }
00351
00352         GPIO_PRINT( "SPI Mode : " << std::hex << (int)(m_spi_mode) );
00353         GPIO_PRINT( "SPI Bits Per Word : " << std::dec << (int)(
m_spi_bits) );
00354         GPIO_PRINT( "SPI Max Speed : " << std::dec << m_spi_speed );
00355         GPIO_PRINT( "SPI Delay : " << std::dec << m_spi_delay );
00356         GPIO_PRINT( "SPI Opened" );
00357     }
00358
00359     //=====
00360     //=====
00361
00362     // Close SPI Channel
00363     void Beagle_GPIO::closeSPI()
00364     {
00365         if ( m_spi_fd >= 0)
00366         {
00367             GPIO_PRINT( "Closing SPI Device" );
00368             close( m_spi_fd );
00369             delete [] m_spi_buffer_rx;
00370         }
00371     }
00372
00373     //=====
00374     //=====
00375
00376     // Send SPI Buffer
00377     void Beagle_GPIO::sendSPIBuffer( unsigned long _buffer, int _size )

```



```

00378 {
00379     gp_assert( m_spi_fd >= 0 );
00380     gp_assert( _buffer > 0 );
00381     gp_assert( _size > 0 );
00382
00383     m_spi_ioc_tr.tx_buf = _buffer;
00384     m_spi_ioc_tr.rx_buf = (unsigned long)(m_spi_buffer_rx);
00385     m_spi_ioc_tr.len = _size;
00386     m_spi_ioc_tr.delay_usecs = m_spi_delay;
00387     m_spi_ioc_tr.speed_hz = m_spi_speed;
00388     m_spi_ioc_tr.bits_per_word = m_spi_bits;
00389
00390     if ( ioctl( m_spi_fd, SPI_IOC_MESSAGE(1), &m_spi_ioc_tr ) < 1 )
00391     {
00392         GPIO_ERROR( "Cannot send SPI Buffer, size=" <<
std::dec << _size );
00393         return;
00394     }
00395 }
00396
00397 //=====
00398 //=====
00399
00400

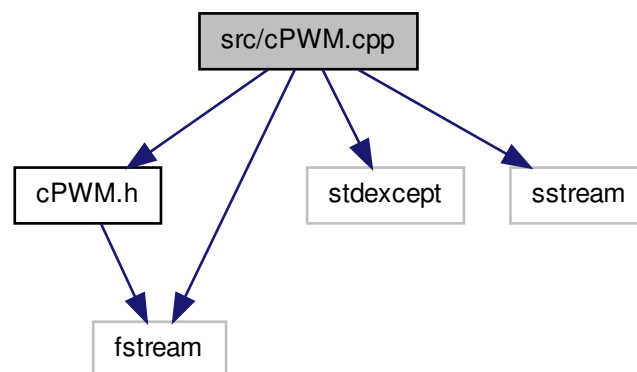
```

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

```

00001 // $Id$
00011 // $Log$
00012
00014
00015 #include "cPWM.h"
00016 #include <stdexcept>
00017 #include <fstream>
00018 #include <sstream>
00019
00033 cPWM::cPWM(int id)
00034 {
00037     cPWM::id = id;
00038
00039     std::stringstream sysfsfile_dutyA_ns;
00040     std::stringstream sysfsfile_dutyA_percent;
00041
00042     std::stringstream sysfsfile_dutyB_ns;
00043     std::stringstream sysfsfile_dutyB_percent;
00044
00045     std::stringstream sysfsfile_period_ns;
00046     std::stringstream sysfsfile_period_freq;
00047
00048     std::stringstream sysfsfile_polarityA;
00049     std::stringstream sysfsfile_runA;
00050     std::stringstream sysfsfile_requestA;
00051
00052     std::stringstream sysfsfile_polarityB;
00053     std::stringstream sysfsfile_runB;
00054     std::stringstream sysfsfile_requestB;
00055
00056     sysfsfile_dutyA_ns << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A <<
    "/" << SYSFS_EHRPWM_DUTY_NS;
00057     sysfsfile_dutyA_percent << SYSFS_EHRPWM_PREFIX << id <<
    SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_DUTY_PERCENT;
00058
00059     sysfsfile_dutyB_ns << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_B <<
    "/" << SYSFS_EHRPWM_DUTY_NS;
00060     sysfsfile_dutyB_percent << SYSFS_EHRPWM_PREFIX << id <<
    SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_DUTY_PERCENT;
00061
00062     sysfsfile_period_ns << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A <
    < "/" << SYSFS_EHRPWM_PERIOD_NS;
00063     sysfsfile_period_freq << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_A
    << "/" << SYSFS_EHRPWM_PERIOD_FREQ;
00064
00065     sysfsfile_polarityA << SYSFS_EHRPWM_PREFIX << id <<
    SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_POLARITY;
00066     sysfsfile_runA << SYSFS_EHRPWM_PREFIX << id <<
    SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_RUN;
00067     sysfsfile_requestA << SYSFS_EHRPWM_PREFIX << id <<
    SYSFS_EHRPWM_SUFFIX_A << "/" << SYSFS_EHRPWM_REQUEST;
00068
00069     sysfsfile_polarityB << SYSFS_EHRPWM_PREFIX << id << SYSFS_EHRPWM_SUFFIX_B <
    < "/" << SYSFS_EHRPWM_POLARITY;
00070     sysfsfile_runB << SYSFS_EHRPWM_PREFIX << id <<
    SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_RUN;
00071     sysfsfile_requestB << SYSFS_EHRPWM_PREFIX << id <<
    SYSFS_EHRPWM_SUFFIX_B << "/" << SYSFS_EHRPWM_REQUEST;
00072
00073     sysfsfid_dutyA_ns.open(sysfsfile_dutyA_ns.str().c_str());
00074     sysfsfid_dutyA_percent.open(sysfsfile_dutyA_percent.str().c_str());

```

```

00075
00076     sysfsfid_dutyB_ns.open(sysfsfile_dutyB_ns.str().c_str());
00077     sysfsfid_dutyB_percent.open(sysfsfile_dutyB_percent.str().c_str());
00078
00079     sysfsfid_period_ns.open(sysfsfile_period_ns.str().c_str());
00080     sysfsfid_period_freq.open(sysfsfile_period_freq.str().c_str());
00081
00082         sysfsfid_polarityA.open(sysfsfile_polarityA.str().c_str());
00083         sysfsfid_runA.open(sysfsfile_runA.str().c_str());
00084
00085     sysfsfid_requestA.open(sysfsfile_requestA.str().c_str());
00086         sysfsfid_polarityB.open(sysfsfile_polarityB.str().c_str());
00087
00088     sysfsfid_runB.open(sysfsfile_runB.str().c_str());
00089     sysfsfid_requestB.open(sysfsfile_requestB.str().c_str());
00090 }
00091
00092 void cPWM::DutyA_ns(unsigned int nanoseconds)
00093 {
00094     if(nanoseconds > cPWM::period)
00095         throw std::out_of_range("DutyA_ns: ");
00096
00097     cPWM::dutyA = nanoseconds;
00098     sysfsfid_dutyA_ns << nanoseconds << std::endl;
00099 }
00100
00101 void cPWM::DutyA_percent(unsigned int percent)
00102 {
00103     if(percent > 100)
00104         throw std::out_of_range("DutyA_percent: ");
00105
00106     sysfsfid_dutyA_percent << percent << std::endl;
00107 }
00108
00109 void cPWM::DutyB_ns(unsigned int nanoseconds)
00110 {
00111     if(nanoseconds > cPWM::period)
00112         throw std::out_of_range("DutyB_ns: ");
00113
00114     cPWM::dutyB = nanoseconds;
00115     sysfsfid_dutyB_ns << nanoseconds << std::endl;
00116 }
00117
00118 void cPWM::DutyB_percent(unsigned int percent)
00119 {
00120     if(percent > 100)
00121         throw std::out_of_range("DutyB_percent: ");
00122
00123     sysfsfid_dutyB_percent << percent << std::endl;
00124 }
00125
00126 void cPWM::Period_ns(unsigned int nanoseconds)
00127 {
00128     cPWM::period = nanoseconds;
00129     cPWM::freq_Hz = 1000000000 / nanoseconds;
00130     sysfsfid_period_ns << nanoseconds << std::endl;
00131 }
00132
00133 void cPWM::Period_freq(unsigned int freq_Hz)
00134 {
00135     cPWM::freq_Hz = freq_Hz;
00136     cPWM::period = 1000000000 / freq_Hz;
00137     sysfsfid_period_freq << freq_Hz << std::endl;
00138 }
00139
00140 void cPWM::PolarityA(Polarity polarity)
00141 {
00142     switch (polarity)
00143     {
00144     case ActiveHigh: sysfsfid_polarityA << 1 << std::endl;
00145                     break;
00146     case ActiveLow:  sysfsfid_polarityA << 0 << std::endl;
00147     }
00148 }

```

```

00191             break;
00192         }
00193         cPWM::polarityA = polarity;
00194     }
00195
00201 void cPWM::RunA()
00202 {
00203     sysfsfid_runA << "1" << std::endl;
00204     cPWM::runA = 1;
00205 }
00206
00212 void cPWM::StopA()
00213 {
00214     sysfsfid_runA << "0" << std::endl;
00215     cPWM::runA = 0;
00216 }
00217
00224 void cPWM::PolarityB(Polarity polarity)
00225 {
00226     switch (polarity)
00227     {
00228     case ActiveHigh: sysfsfid_polarityB << 1 << std::endl;
00229                     break;
00230     case ActiveLow: sysfsfid_polarityB << 0 << std::endl;
00231                     break;
00232     }
00233     cPWM::polarityB = polarity;
00234 }
00235
00241 void cPWM::RunB()
00242 {
00243     cPWM::runB = 1;
00244     sysfsfid_runB << "1" << std::endl;
00245 }
00246
00251 void cPWM::StopB()
00252 {
00253     cPWM::runB = 0;
00254     sysfsfid_runB << "0" << std::endl;
00255 }
00256
00261 cPWM::~cPWM()
00262 {
00263     sysfsfid_runA << "0" << std::endl;
00264
00265     sysfsfid_runB << "0" << std::endl;
00266 }

```

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

```

00001
00012 #include <stdint.h>
00013 #include <iostream>
00014 #include <iomanip>
00015 #include <stdexcept>
00016
00017 #include <sys/time.h>
00018
00019 #include "gx3communicator.h"
00020 using namespace USU;
00021
00022 #include "messages.h"
00023
00024 //int timeval_subtract (struct timeval * result, struct timeval * x, struct
    timeval * y)
00025 //{
00026 //    /* Perform the carry for the later subtraction by updating y. */
00027 //    if (x->tv_usec < y->tv_usec) {
00028 //        int nsec = (y->tv_usec - x->tv_usec) / 1000000 + 1;
00029 //        y->tv_usec -= 1000000 * nsec;
00030 //        y->tv_sec += nsec;
00031 //    }
00032 //    if (x->tv_usec - y->tv_usec > 1000000) {
00033 //        int nsec = (x->tv_usec - y->tv_usec) / 1000000;
00034 //        y->tv_usec += 1000000 * nsec;
00035 //        y->tv_sec -= nsec;
00036 //    }
00037
00038 //    /* Compute the time remaining to wait.
00039 //       tv_usec is certainly positive. */
00040 //    result->tv_sec = x->tv_sec - y->tv_sec;
00041 //    result->tv_usec = x->tv_usec - y->tv_usec;
00042
00043 //    /* Return 1 if result is negative. */
00044 //    return x->tv_sec < y->tv_sec;
00045 //}
00046
00047 GX3Communicator::GX3Communicator(int priority, const char *serialDevice,
    SerialPort::BaudRate baudRate)
00048     :RtThread(priority), mSerialPort(serialDevice), mBaudRate(baudRate),
    mKeepRunning(false)
00049 {
00050
00051 }
00052
00053 void GX3Communicator::initialize()
00054 {
00055     mSerialPort.Open(mBaudRate);
00056     if(mSerialPort.IsOpen() == false)
00057         throw std::runtime_error("Opening SerialPort failed");
00058
00059     /*
00060     Set up the 3DM-GX25 with the following settings (different from
    default):
00061     - Data rate 50 Hz
00062     - Enable little endian for floating points
00063     */
00064     SamplingSettings initSettings(SamplingSettings::Change, 20,
    SamplingSettings::FlagDefault |
00065     SamplingSettings::FlagFloatLittleEndian
00066     | SamplingSettings::FlagEnableQuaternion);
00067

```

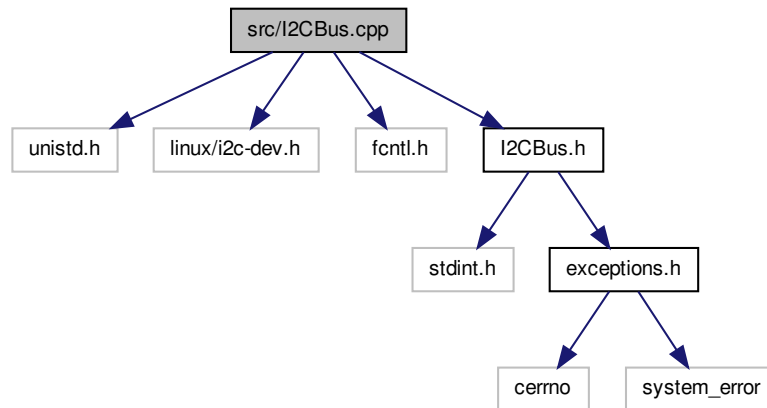
```

00068         if(initSettings.sendCommand(mSerialPort) == false)
00069             throw std::runtime_error("Setting SamplingSettings failed");
00070
00071     }
00072
00073     void GX3Communicator::run()
00074     {
00075
00076         mKeepRunning = true;
00077
00078         // Activate Continuous mode
00079         SetContinuousMode setCont(ACC_ANG_MAG_VEC);
00080         if(setCont.sendCommand(mSerialPort) == false)
00081             std::cerr << " Set continuous mode failed " << std::endl;
00082
00083         // struct timeval start, now, elapsed;
00084
00085         // gettimeofday(&start, NULL);
00086         while(mKeepRunning)
00087         {
00088             packet_ptr data(new AccAngMag);
00089             if(data->readFromSerial(mSerialPort))
00090             {
00091                 mQueue.push(data);
00092                 // gettimeofday(&now, NULL);
00093                 // timeval_subtract(&elapsed, &now, &start);
00094                 // unsigned long long timestamp = elapsed.tv_sec * 1000 +
00095                 // elapsed.tv_usec / 1000; // in ms since start
00096                 // std::cout << (*data) << std::endl;
00097             }
00098             else
00099             {
00100                 std::cout << "readFromSerial failed" << std::endl;
00101                 // throw std::runtime_error("Getting PackageData
00102                 // failed"); /// TODO: Error?
00103             }
00104
00105             std::cerr << "GX3COMMUNICATOR: Got signal to terminate" << std::endl;
00106             std::cerr << "GX3COMMUNICATOR: Stopping IMU continuous mode..." <<
std::endl;
00107             // Stop continuous mode
00108             setCont.mCommand[3] = 0;
00109             if(setCont.sendCommand(mSerialPort) == false)
00110                 ;
00111
00112             std::cerr << "GX3COMMUNICATOR: IMU continuous mode stopped" << std::endl;
00113             std::cerr << "GX3COMMUNICATOR: Terminating now..." << std::endl;
00114         }
00115
00116

```

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

```

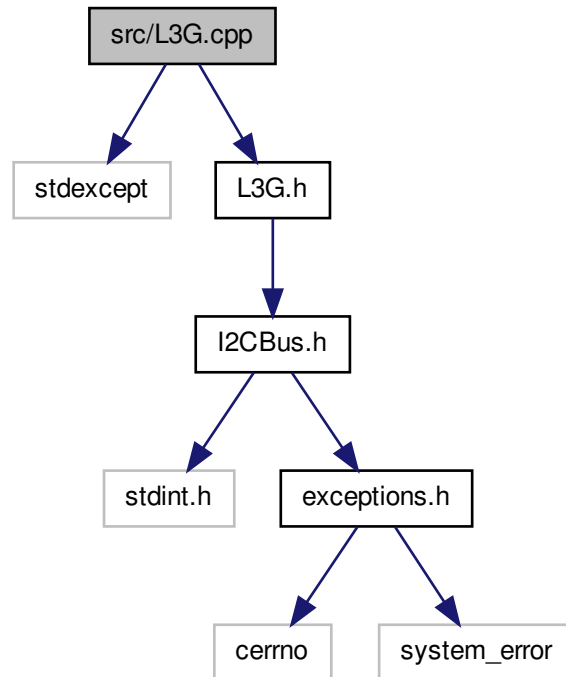
00001 // #include <sys/ioctl.h>
00002 #include <unistd.h>
00003 #include <linux/i2c-dev.h>
00004 #include <fcntl.h>
00005
00006 #include "I2CBus.h"
00007
00008 I2CBus::I2CBus(const char * deviceName)
00009 {
00010     fd = open(deviceName, O_RDWR);
00011     if (fd == -1)
00012     {
00013         throw posix_error("Failed to open I2C device.");
00014     }
00015 }
00016
00017 I2CBus::~I2CBus()
00018 {
00019     close(fd);
00020 }
00021
00022 void I2CBus::addressSet(uint8_t address)
00023 {
00024     int result = ioctl(fd, I2C_SLAVE, address);
00025     if (result == -1)
00026     {
00027         throw posix_error("Failed to set address.");
00028     }
00029 }
00030
00031 void I2CBus::writeByte(uint8_t command, uint8_t data)
00032 {
00033     int result = i2c_smbus_write_byte_data(fd, command, data);

```

```
00034     if (result == -1)
00035     {
00036         throw posix_error("Failed to write byte to I2C.");
00037     }
00038 }
00039
00040 void I2CBus::writeByte(uint8_t data)
00041 {
00042     int result = i2c_smbus_write_byte(fd, data);
00043     if (result == -1)
00044     {
00045         throw posix_error("Failed to write raw byte to I2C.");
00046     }
00047 }
00048
00049 uint8_t I2CBus::readByte(uint8_t command)
00050 {
00051     int result = 0;
00052     result = i2c_smbus_read_byte_data(fd, command);
00053     if (result == -1)
00054     {
00055         // throw posix_error("Failed to read byte from I2C.");
00056         return 0;
00057     }
00058     return (uint8_t) result;
00059 }
00060
00061 uint8_t I2CBus::readByte()
00062 {
00063     int result = i2c_smbus_read_byte(fd);
00064     if (result == -1)
00065     {
00066         throw posix_error("Failed to read raw byte from I2C.");
00067     }
00068     return result;
00069 }
00070
00071 uint16_t I2CBus::readWord(uint8_t command)
00072 {
00073     int result = i2c_smbus_read_word_data(fd, command);
00074     if (result == -1)
00075     {
00076         throw posix_error("Failed to read word from I2C.");
00077     }
00078     return result;
00079 }
00080
00081 uint16_t I2CBus::readWord()
00082 {
00083     uint16_t temp;
00084     int result = read(fd, &temp, sizeof(uint16_t));
00085     if (result == -1)
00086     {
00087         throw posix_error("Failed to read raw word from I2C");
00088     }
00089     return temp;
00090 }
00091
00092 int I2CBus::tryReadByte(uint8_t command)
00093 {
00094     return i2c_smbus_read_byte_data(fd, command);
00095 }
00096
00097
00098 void I2CBus::readBlock(uint8_t command, uint8_t size, uint8_t * data)
00099 {
00100     int result = i2c_smbus_read_i2c_block_data(fd, command, size, data);
00101     if (result != size)
00102     {
00103         throw posix_error("Failed to read block from I2C.");
00104     }
00105 }
```


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](#).

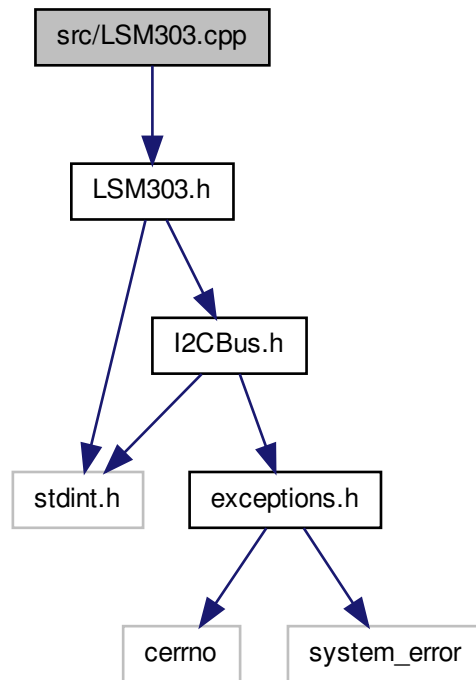
9.52 src/L3G.cpp

```
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);

```

```

00059         // Only the DLHC should be responding on the high address.
00060         device = Device::LSM303DLHC;
00061     }
00062     else
00063     {
00064         // Only the DLM has a LSM303_WHO_AM_I_M register.
00065         device = i2c_mag.tryReadByte(LSM303_WHO_AM_I_M) == 0x3C ?
Device::LSM303DLM : Device::LSM303DLH;
00066     }
00067
00068     // Make sure to throw an exception if we don't have the right address.
00069     readAccReg(LSM303_CTRL_REG1_A);
00070
00071     if (readMagReg(LSM303_WHO_AM_I_M) != 0x3C)
00072     {
00073         throw std::runtime_error("LSM303: Error getting \"Who Am I\" register.
\n");
00074     }
00075
00076     // Enable accelerometer.
00077     if (device == Device::LSM303DLHC)
00078     {
00079         writeAccReg(LSM303_CTRL_REG1_A, 0b01000111); // Normal power mode, all
axes enabled, 50 Hz
00080         writeAccReg(LSM303_CTRL_REG4_A, 0b10001000); // 2 g full scale: FS = 00
on DLHC, high resolution output mode
00081     }
00082     else
00083     {
00084         writeAccReg(LSM303_CTRL_REG1_A, 0b00100111); // normal power mode, all
axes enabled, 50 Hz
00085         writeAccReg(LSM303_CTRL_REG4_A, 0b10000000); // 2 g full scale: FS = 00
on DLH, DLM
00086     }
00087
00088     // Enable magnetometer
00089     // Continuous conversion mode
00090     writeMagReg(LSM303_CRB_REG_M, 0b01000000); // FS = +-1.9 gauss
00091     writeMagReg(LSM303_MR_REG_M, 0x00);
00092 }
00093
00094 void LSM303::readAcc(void)
00095 {
00096     uint8_t block[6];
00097     i2c_acc.readBlock(0x80 | LSM303_OUT_X_L_A, sizeof(block), block);
00098
00099     a[0] = (int16_t)(block[0] | block[1] << 8) >> 4;
00100     a[1] = (int16_t)(block[2] | block[3] << 8) >> 4;
00101     a[2] = (int16_t)(block[4] | block[5] << 8) >> 4;
00102 }
00103
00104 void LSM303::readMag(void)
00105 {
00106     uint8_t block[6];
00107     i2c_mag.readBlock(0x80 | LSM303_OUT_X_H_M, sizeof(block), block);
00108
00109     // DLM, DLHC: register address order is X,Z,Y with high bytes first
00110     m[0] = (int16_t)(block[1] | block[0] << 8);
00111     m[1] = (int16_t)(block[5] | block[4] << 8);
00112     m[2] = (int16_t)(block[3] | block[2] << 8);
00113
00114     // TODO: handle DLH properly here (switch two components?)
00115 }
00116
00117 // Reads all 6 channels of the LSM303 and stores them in the object variables
00118 void LSM303::read(void)
00119 {
00120     readAcc();
00121     readMag();
00122 }
00123

```

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/CommandLine.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
```

```

00031
00032 MainThread kalmanFilter(5, 20000 , "/dev/i2c-2", "/dev/i2c-3");
00033 //bool run = true;
00034
00035 void endProgram(int s)
00036 {
00037     std::cerr << "MAIN: Got signal for termination" << std::endl;
00038     std::cerr << "MAIN: Stopping kalman filter thread..." << std::endl;
00039     kalmanFilter.stop();
00040 }
00041
00042
00043 int main(int argc, char **argv)
00044 {
00045     // Register endProgram function as
00046     // signal handler for the kill signal (ctrl+c)
00047     struct sigaction sigIntHandler;
00048     sigIntHandler.sa_handler = endProgram;
00049     sigemptyset(&sigIntHandler.sa_mask);
00050     sigIntHandler.sa_flags = 0;
00051
00052     sigaction(SIGINT, &sigIntHandler, NULL);
00053
00054     try
00055     {
00056         // Register commandline options to parser
00057         cmd.add(trajFile);
00058         cmd.add(pgain);
00059         cmd.add(mode);
00060
00061         cmd.parse(argc, argv);
00062
00063         // Evaluate command line options
00064         if(mode.getValue() == "simpleControl")
00065         {
00066             kalmanFilter.initializeModeSimpleControl(trajFile.getValue(), pgain
00067             .getValue());
00068             kalmanFilter.setMode(MainThread::SimpleControl);
00069         }
00070         else if(mode.getValue() == "pololu")
00071         {
00072             kalmanFilter.setMode(MainThread::CollectPololuData);
00073         }
00074         else if(mode.getValue() == "microstrain")
00075         {
00076             kalmanFilter.setMode(MainThread::CollectMicroStrainData);
00077         }
00078         else if(mode.getValue() == "collect")
00079         {
00080             kalmanFilter.setMode(MainThread::CollectData);
00081         }
00082         else
00083         {
00084             throw std::runtime_error("MAIN: Unknown mode selected. Terminating"
00085             );
00086             return 1;
00087         }
00088         kalmanFilter.start();
00089         if(kalmanFilter.join() )
00090         {
00091             std::cerr << "MAIN: Kalman filter thread joined" << std::endl;
00092             std::cerr << "MAIN: Terminating now..." << std::endl;
00093             return 0;
00094         }
00095         else
00096         {
00097             std::cerr << "MAIN: Joining Kalman filter thread failed" <<
00098             std::endl;
00099             std::cerr << "MAIN: Terminating now..." << std::endl;
00100             return 1;
00101         }
00102     } catch (TCLAP::ArgException &e) // catch any exceptions

```



```

00102     {
00103         std::cerr << "error: " << e.error() << " for arg " << e.argId() <<
        std::endl;
00104         return 1;
00105     }
00106
00107 }
```

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

```

00001
00012 #include<iostream>
00013 using std::cout;
00014 using std::endl;
00015
00016 #include <sys/time.h>
00017 #include <unistd.h>
00018
00019
00020 #include "mainthread.h"
00021 #include "vector.h"
```

```

00022
00023 using namespace USU;
00024
00025
00026 int timeval_subtract (struct timeval * result, struct timeval * x, struct
    timeval * y)
00027 {
00028     /* Perform the carry for the later subtraction by updating y. */
00029     if (x->tv_usec < y->tv_usec) {
00030         int nsec = (y->tv_usec - x->tv_usec) / 1000000 + 1;
00031         y->tv_usec -= 1000000 * nsec;
00032         y->tv_sec += nsec;
00033     }
00034     if (x->tv_usec - y->tv_usec > 1000000) {
00035         int nsec = (x->tv_usec - y->tv_usec) / 1000000;
00036         y->tv_usec += 1000000 * nsec;
00037         y->tv_sec -= nsec;
00038     }
00039
00040     /* Compute the time remaining to wait.
00041        tv_usec is certainly positive. */
00042     result->tv_sec = x->tv_sec - y->tv_sec;
00043     result->tv_usec = x->tv_usec - y->tv_usec;
00044
00045     /* Return 1 if result is negative. */
00046     return x->tv_sec < y->tv_sec;
00047 }
00048
00049 MainThread::MainThread(int priority, unsigned int period_us, const char *i2cImu
    , const char *i2cMotor)
00050 :PeriodicRtThread(priority, period_us), mMode(CollectPololuData), mImu(
    i2cImu),
00051 mGX3(priority, "/dev/ttyO4"), mKeepRunning(false)
00052 {
00053 }
00054
00055 void MainThread::run()
00056 {
00057     switch (mMode)
00058     {
00059     case SimpleControl:      runSimpleControl();
00060                             break;
00061     case CollectPololuData:  runCollectPololu();
00062                             break;
00063     case CollectMicroStrainData: runCollectMicroStrain();
00064                             break;
00065     case CollectData:        runCollectBoth();
00066                             break;
00067     }
00068
00069     std::cerr << "KALMANFILTER: Terminating now..." << std::endl;
00070 }
00071
00072 bool MainThread::getState()
00073 {
00074     ScopedLock scLock(mStateLock);
00075     return mState;
00076 }
00077
00078 void MainThread::initializeModeSimpleControl(std::string trajFilename, float
    pgain)
00079 {
00080     std::ifstream inFile;
00081     inFile.open(trajFilename);
00082     if(!inFile.is_open())
00083         throw std::runtime_error("MotorController: Could not open input file");
00084
00085     Command temp;
00086     while(true)
00087     {
00088         inFile >> temp.time;
00089
00090         float x,y,z;
00091         inFile >> x >> y >> z;

```

```

00092         temp.angVel << x, y, z;
00093
00094         if(inFile.eof())
00095             break;
00096         mCommandList.push_back(temp);
00097     }
00098     inFile.close();
00099
00100     cout << "Read " << mCommandList.size() << " commands." << endl;
00101
00102     mMotors.setPGain(pgain);
00103 }
00104
00105 void MainThread::runSimpleControl()
00106 {
00107     vector gyro;
00108     mKeepRunning = true;
00109     struct timeval start, now, elapsed;
00110
00111     if(mCommandList.empty())
00112     {
00113         std::cerr << "Error: No command list loaded. Terminating";
00114         return;
00115     }
00116
00117     // mImu.enable();
00118
00119     std::vector<Command>::const_iterator commandIt = mCommandList.begin();
00120     mMotors.setSetValue(commandIt->angVel);
00121     int countdown = commandIt->time;
00122
00123     gettimeofday(&start, NULL);
00124     unsigned lastTime = 0;
00125     waitPeriod();
00126
00127     while(mKeepRunning)
00128     {
00129         gettimeofday(&now, NULL);
00130
00131         // gyro = mImu.readGyro();
00132
00133         // Run countdown
00134         timeval_subtract(&elapsed, &now, &start);
00135         unsigned time = elapsed.tv_sec * 1000 + elapsed.tv_usec / 1000; // in
00136         ms since start
00137         countdown -= time - lastTime;
00138         lastTime = time;
00139
00140         // if countdown over execute next command from list
00141         if(countdown <=0)
00142         {
00143             commandIt++;
00144             // if at the end of the commandList start again from the beginning
00145             if(commandIt == mCommandList.end())
00146                 commandIt = mCommandList.begin();
00147
00148             countdown = commandIt->time;
00149             mMotors.setSetValue(commandIt->angVel);
00150         }
00151
00152         // Always use mutex, when changing state
00153         mMotors.controlFromGyro(gyro);
00154
00155         waitPeriod();
00156     }
00157
00158     gyro(2) = 1;
00159     mMotors.setSetValue(gyro);
00160     mMotors.controlFromGyro(gyro);
00161     waitPeriod();
00162     gyro(2) = 0;
00163     mMotors.setSetValue(gyro);

```

```

00165     mMotors.controlFromGyro(gyro);
00166
00167     std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
00168 }
00169
00170 void MainThread::runCollectPololu()
00171 {
00172     vector acc, mag, gyro;
00173     mKeepRunning = true;
00174     struct timeval start, now, elapsed;
00175
00176     mImu.enable();
00177
00178     gettimeofday(&start, NULL);
00179     waitPeriod();
00180
00181     // Create an object to set the output format for the vectors
00182     Eigen::IOFormat csv(Eigen::StreamPrecision, Eigen::DontAlignCols, " ", " ",
00183
00184 );
00185
00186 while(mKeepRunning)
00187 {
00188     gettimeofday(&now, NULL);
00189
00190     acc = mImu.readAcc();
00191     mag = mImu.readMag();
00192     gyro = mImu.readGyro();
00193
00194     timeval_subtract(&elapsed, &now, &start);
00195     unsigned time = elapsed.tv_sec * 1000 + elapsed.tv_usec / 1000; // in
00196 ms since start
00197
00198     // print data
00199     cout << time << ",\t" << acc.format(csv) << ",\t" << mag.format(csv) <<
00200 ",\t" << gyro.format(csv) << endl;
00201
00202     waitPeriod();
00203 }
00204
00205 std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
00206 }
00207
00208 void MainThread::runCollectMicroStrain()
00209 {
00210     mKeepRunning = true;
00211
00212     mGX3.initialize();
00213     mGX3.start();
00214     packet_ptr lastState;
00215     while(mKeepRunning)
00216     {
00217         if(mGX3.isEmpty() == false)
00218         {
00219             int length = mGX3.size();
00220             while(length-->1)
00221             {
00222                 mGX3.pop();
00223             }
00224             lastState = mGX3.front();
00225             mGX3.pop();
00226         }
00227
00228         cout << (*lastState) << endl;
00229         waitPeriod();
00230     }
00231
00232     std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
00233     std::cerr << "KALMANFILTER: Stopping Gx3-communicator..." << std::endl;
00234     mGX3.stop();
00235     if(mGX3.join() )

```

```

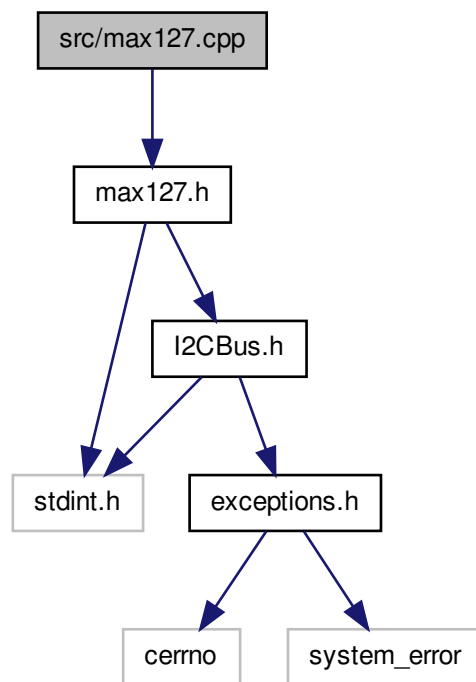
00236     {
00237         std::cerr << "KALMANFILTER: Gx3-communicator joined" << std::endl;
00238     }
00239     else
00240     {
00241         std::cerr << "KALMANFILTER: Joining Gx3-communicator failed" <<
std::endl;
00242     }
00243 }
00244
00245 void MainThread::runCollectBoth()
00246 {
00247     vector acc, mag, gyro;
00248     mKeepRunning = true;
00249     struct timeval start, now, elapsed;
00250
00251     mImu.enable();
00252     mGX3.initialize();
00253     mGX3.start();
00254
00255     gettimeofday(&start, NULL);
00256     waitPeriod();
00257
00258     // Create an object to set the output format for the vectors
00259     Eigen::IOFormat csv(Eigen::StreamPrecision, Eigen::DontAlignCols, " ", " ", "
");
00260
00261     while(mKeepRunning)
00262     {
00263         gettimeofday(&now, NULL);
00264
00265         acc = mImu.readAcc();
00266         mag = mImu.readMag();
00267         gyro = mImu.readGyro();
00268
00269         timeval_subtract(&elapsed, &now, &start);
00270         unsigned time = elapsed.tv_sec * 1000 + elapsed.tv_usec / 1000; // in
ms since start
00271
00272         packet_ptr lastState;
00273
00274         if(mGX3.isEmpty() == false)
00275         {
00276             int length = mGX3.size();
00277             while(length-->1)
00278             {
00279                 mGX3.pop();
00280             }
00281
00282             lastState = mGX3.front();
00283             mGX3.pop();
00284         }
00285
00286         // print data
00287         cout << (*lastState) << "\t" << time << ",\t" << acc.format(csv) << "
\t" << mag.format(csv) << ",\t" << gyro.format(csv) << endl;
00288
00289         waitPeriod();
00290     }
00291
00292     std::cerr << "KALMANFILTER: Got signal to terminate" << std::endl;
00293     std::cerr << "KALMANFILTER: Stopping Gx3-communicator..." << std::endl;
00294     mGX3.stop();
00295     if(mGX3.join() )
00296     {
00297         std::cerr << "KALMANFILTER: Gx3-communicator joined" << std::endl;
00298     }
00299     else
00300     {
00301         std::cerr << "KALMANFILTER: Joining Gx3-communicator failed" <<
std::endl;
00302     }
00303 }
00304 MainThread::Mode MainThread::getMode() const

```

```
00305 {  
00306     return mMode;  
00307 }  
00308  
00309 void MainThread::setMode(const Mode &value)  
00310 {  
00311     mMode = value;  
00312 }  
00313
```

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

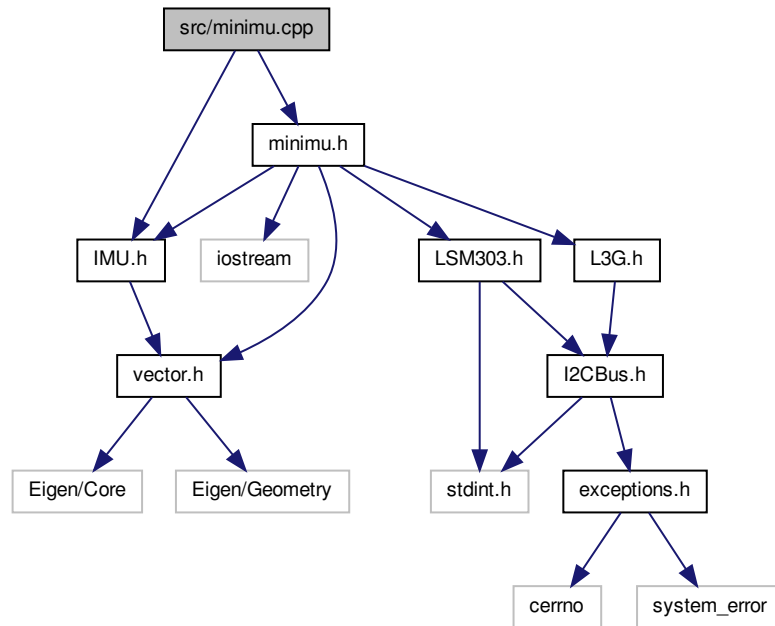
Definition in file [max127.cpp](#).

9.60 src/max127.cpp

```
00001
00011 #include "max127.h"
00012 using namespace USU;
00013
00014 Max127::Max127(const char *i2cdevice)
00015     :mI2c(i2cdevice)
00016 {
00017     mI2c.addressSet(I2C_ADDRESS);
00018 }
00019
00020 int16_t Max127::readRaw(uint8_t channel)
00021 {
00022     // Compile the full control byte by setting the channel bits
00023     mI2c.writeByte(CONTROL_BYTE | (channel << SEL0) );
00024
00025     uint16_t rawValue = mI2c.readWord();
00026
00027     // From the read word use the high byte as low byte and vice versa
00028     // Then move all bits 4 to the left because it is only a 12 bit number.
00030     return ( (int16_t) ( ( (rawValue & 0xFF00)>>8) | ( (rawValue & 0x00FF) <<8)
00031 ) >> 4 );
00032
00033 }
00034
00035 float Max127::readVoltage(unsigned int channel)
00036 {
00037     // fullscale = +-5V --> resolution = 10V/4096
00038     const float scaleVoltage = 2.4414063e-3f;
00039     return (readRaw(channel) * scaleVoltage);
00040 }
```

9.61 src/minimu.cpp File Reference

#include "minimu.h" #include "IMU.h" Include dependency graph for minimu.cpp:



9.62 src/minimu.cpp

```

00001 #include "minimu.h"
00002 using namespace USU;
00003 #include "IMU.h"
00004
00005 MinImu::MinImu(const char *i2cDeviceName)
00006     :compass(i2cDeviceName), gyro(i2cDeviceName)
00007 {
00008 }
00009 }
00010
00011 void MinImu::enable()
00012 {
00013     compass.enable();
00014     gyro.enable();
00015 }
00016
00017 vector MinImu::readGyro()
00018 {
00019     // At the full-scale=250 dps setting, the gyro datasheet says
00020     // we get 8.75 mdps/digit.
00021     const float gyro_scale = 0.00875; // in °/s
00022
00023     gyro.read();

```



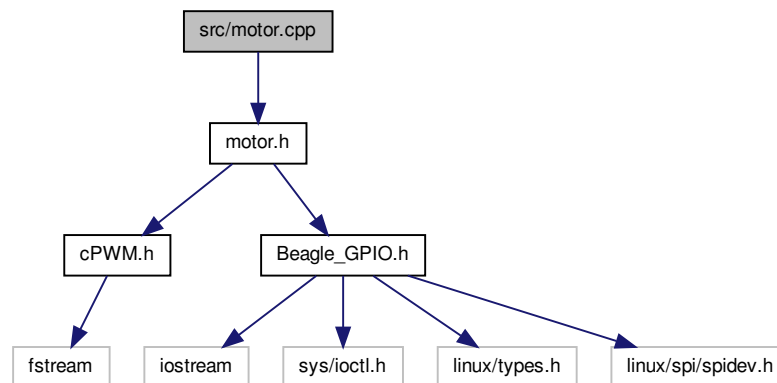
```

00024     IMU::raw_g = int_vector_from_ints(&gyro.g);
00025     return ( vector_from_ints(&gyro.g) * gyro_scale);
00026 }
00027
00028 vector MinImu::readAcc()
00029 {
00030     // LSM303 accelerometer: At 2 g sensitivity, the datasheet says
00031     // we get 1 mg/digit.
00032     const float accel_scale = 0.0010;
00033
00034     compass.readAcc();
00035     IMU::raw_a = int_vector_from_ints(&compass.a);
00036     return vector_from_ints(&compass.a) * accel_scale;
00037 }
00038
00039 vector MinImu::readMag()
00040 {
00041     compass.readMag();
00042     IMU::raw_m = int_vector_from_ints(&compass.m);
00043
00044     return vector_from_ints(&compass.m);
00045 }

```

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

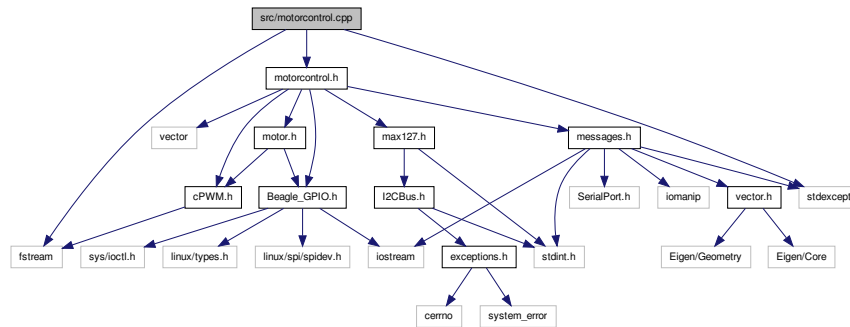
```

00001
00011 #include "motor.h"
00012 using namespace USU;
00013
00014 Motor::Motor(Beagle_GPIO &beagleGpio, Beagle_GPIO::Pins clockwise,
00015             Beagle_GPIO::Pins counterClockwise, cPWM &pwm, SetDutyCycle dutyCycle)
00016 :mBeagleGpio(beagleGpio), mClockwise(clockwise), mCounterClockwise(
00017 counterClockwise),
00018 mPwm(pwm), mSetDutyCycle(dutyCycle), mSpeed(0)
00019 {
00020     mBeagleGpio.configurePin(mClockwise, Beagle_GPIO::kOUTPUT);
00021     mBeagleGpio.enablePinInterrupts( mClockwise, false );
00022     mBeagleGpio.writePin(mClockwise, 0);
00023
00024     mBeagleGpio.configurePin(mCounterClockwise, Beagle_GPIO::kOUTPUT);
00025     mBeagleGpio.enablePinInterrupts( mCounterClockwise, false );
00026     mBeagleGpio.writePin(mCounterClockwise, 0);
00027
00028     setSpeed(mSpeed);
00029 }
00030 void Motor::setSpeed(int speed)
00031 {
00032     if (speed > 0)
00033     {
00034         // Make sure speed <100
00035         speed = speed<100 ? speed : 99;
00036         mBeagleGpio.writePin(mClockwise, 1);
00037         mBeagleGpio.writePin(mCounterClockwise, 0);
00038         (mPwm.*mSetDutyCycle)(speed);
00039         mSpeed = speed;
00040     }
00041     else if (speed < 0)
00042     {
00043         // Make sure speed >-100
00044         speed = speed>-100 ? speed : -99;
00045         mBeagleGpio.writePin(mClockwise, 0);
00046         mBeagleGpio.writePin(mCounterClockwise, 1);
00047         (mPwm.*mSetDutyCycle)(-speed);
00048         mSpeed = speed;
00049     }
00050     else
00051     {
00052         mBeagleGpio.writePin(mClockwise, 0);
00053         mBeagleGpio.writePin(mCounterClockwise, 0);
00054         (mPwm.*mSetDutyCycle)(9);
00055         mSpeed = 0;
00056     }
00057 }

```

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

```

00001
00012 #include <fstream>
00013 #include <stdexcept>
00014
00015 #include "motorcontrol.h"
00016 using namespace USU;
00017
00018 MotorControl::MotorControl(const char *i2cDevice)
00019     :mPwm1(1), mPwm2(2), mAnalog(i2cDevice), mPGain(1.0)
00020 {
00021     // Inititalize the four motors
00022     mMotor[0] = new Motor(mBeagleGpio, Beagle_GPIO::P8_31, Beagle_GPIO::P8_29,
00023         mPwm1, &cPWM::DutyA_percent);
00023     mMotor[1] = new Motor(mBeagleGpio, Beagle_GPIO::P8_27, Beagle_GPIO::P8_25,
00024         mPwm1, &cPWM::DutyB_percent);
00024     mMotor[2] = new Motor(mBeagleGpio, Beagle_GPIO::P8_23, Beagle_GPIO::P8_21,
00025         mPwm2, &cPWM::DutyA_percent);
00025     mMotor[3] = new Motor(mBeagleGpio, Beagle_GPIO::P8_18, Beagle_GPIO::P8_17,
00026         mPwm2, &cPWM::DutyB_percent);
00026     mPwm1.Period_freq(100);
00027     mPwm2.Period_freq(100);
00028     mPwm1.RunA();
00029     mPwm1.RunB();
  
```

```

00030     mPwm2.RunA();
00031     mPwm2.RunB();
00032 }
00033
00034 MotorControl::~MotorControl()
00035 {
00036     mPwm1.StopA();
00037     mPwm1.StopB();
00038     mPwm2.StopA();
00039     mPwm2.StopB();
00040 }
00041
00042 void MotorControl::calculateControlResponse(Quaternion state)
00043 {
00044     // mMotor[0]->setSpeed(20);
00045 }
00046
00047 void MotorControl::controlFromGyro(const Eigen::Vector3f & gyro)
00048 {
00049     // float speeds[4];
00050     // float currents[4];
00051     // getAnalog(speeds, currents);
00052     // int speeds_input[4];
00053     // getDutyCycles(speeds_input);
00054     int speed = (int) mSetValue(2);
00055     mMotor[0]->setSpeed(speed);
00056     mMotor[1]->setSpeed(speed);
00057     mMotor[2]->setSpeed(speed);
00058     mMotor[3]->setSpeed(speed);
00059     // mPGain I already have this, do the math to go from speeds to rpms (or
00060     // rad/s), then Eigen::Vector3f err = (gyro - mSetValue)
00061     // From err to 4 * pwms
00062 }
00063
00064 void MotorControl::setMotor(int motor, int dutyCycle)
00065 {
00066     mMotor[motor]->setSpeed(dutyCycle);
00067 }
00068
00069 void MotorControl::getAnalog(int motor, float& aOut1, float& aOut2)
00070 {
00071     aOut1 = mAnalog.readVoltage(motor*2);
00072     aOut2 = mAnalog.readVoltage(motor*2 + 1);
00073 }
00074
00075 void MotorControl::getAnalog(float *aOut1, float *aOut2)
00076 {
00077     aOut1[0] = mAnalog.readVoltage(0);
00078     aOut2[0] = mAnalog.readVoltage(1);
00079     aOut1[1] = mAnalog.readVoltage(2);
00080     aOut2[1] = mAnalog.readVoltage(3);
00081     aOut1[2] = mAnalog.readVoltage(4);
00082     aOut2[2] = mAnalog.readVoltage(5);
00083     aOut1[3] = mAnalog.readVoltage(6);
00084     aOut2[3] = mAnalog.readVoltage(7);
00085 }
00086
00087 void MotorControl::getDutyCycles(int *dc)
00088 {
00089     dc[0] = mMotor[0]->getSpeed();
00090     dc[1] = mMotor[1]->getSpeed();
00091     dc[2] = mMotor[2]->getSpeed();
00092     dc[3] = mMotor[3]->getSpeed();
00093 }
00094
00095 float MotorControl::getPGain() const
00096 {
00097     return mPGain;
00098 }
00099
00100 void MotorControl::setPGain(float value)

```

```

00105 {
00106     mPGain = value;
00107 }
00108 Eigen::Vector3f MotorControl::getSetValue() const
00109 {
00110     return mSetValue;
00111 }
00112
00113 void MotorControl::setSetValue(const Eigen::Vector3f value)
00114 {
00115     mSetValue = value;
00116 }
00117
00118

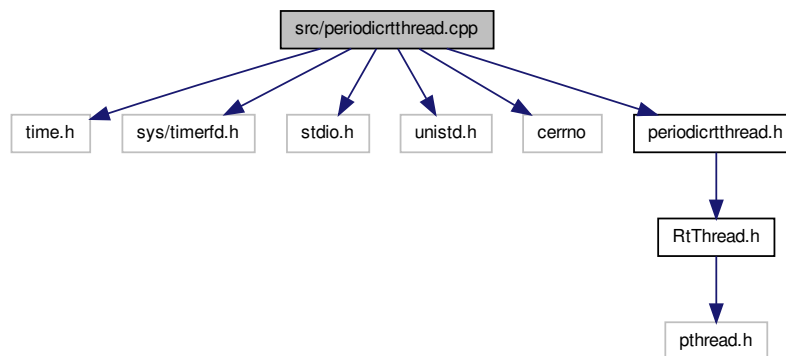
```

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

```

00001

```

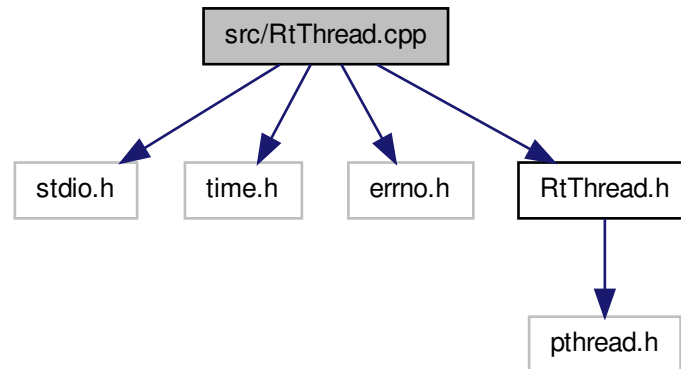
```

00012 #include <time.h>
00013 #include <sys/timerfd.h>
00014 #include <stdio.h>
00015 #include <unistd.h>
00016 #include <cerrno>
00017
00018 #include "periodicrtthread.h"
00019 using namespace USU;
00020
00021
00022 PeriodicRtThread::PeriodicRtThread(int priority, unsigned int period_us)
00023     :RtThread(priority), mMissedWakeUps(0), mPeriod_us(period_us)
00024 {
00025     makeThreadPeriodic();
00026 }
00027
00028
00029 void PeriodicRtThread::makeThreadPeriodic()
00030 {
00031     int ret;
00032     unsigned int ns;
00033     unsigned int sec;
00034     struct itimerspec itval;
00035     /* Create the timer */
00036     if ( (mTimerFd = timerfd_create (CLOCK_MONOTONIC, 0)) == -1)
00037     {
00038         perror("timer_create ");
00039         return;
00040     }
00041
00042     /* Make the timer periodic */
00043     sec = mPeriod_us/1000000;
00044     ns = (mPeriod_us - (sec * 1000000)) * 1000;
00045     itval.it_interval.tv_sec = sec;
00046     itval.it_interval.tv_nsec = ns;
00047     itval.it_value.tv_sec = sec;
00048     itval.it_value.tv_nsec = ns;
00049     if ( (ret = timerfd_settime(mTimerFd, 0, &itval, NULL)) != 0)
00050     {
00051         perror("timerfd_settime ");
00052         return;
00053     }
00054 }
00055
00056 void PeriodicRtThread::waitPeriod()
00057 {
00058     unsigned long long missed;
00059     int ret;
00060
00061     /* Wait for the next timer event. If we have missed any the
00062        number is written to "missed"
00063        While loop to continue the wait if interrupted by a signal.
00064     */
00065     while( (ret = read (mTimerFd, &missed, sizeof (missed)) ) == -1)
00066     {
00067         perror("timer read ");
00068         return;
00069     }
00070
00071     /* "missed" should always be >= 1, but just to be sure, check it is not 0
    anyway */
00072     if (missed > 0)
00073         mMissedWakeUps += (missed - 1);
00074 }

```

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

```
00001
00010 #include <stdio.h>
00011 #include <time.h>
00012 #include <errno.h>
00013
00014 #include "RtThread.h"
00015 using namespace USU;
00016
00017
00018
00019 RtThread::RtThread(int priority):
00020     mPriority(priority), mId(-1), mStarted(false)
00021 {
00022     int ret;
00023     if ( (ret = pthread_attr_init(&mAttr)) != 0)
00024     {
```

```

00025         perror("pthread_attr_init ");
00026         throw "Error";
00027     }
00028     // Set scheduler to (realtime) FIFO
00029     if ( (ret = pthread_attr_setschedpolicy(&mAttr, SCHED_FIFO)) != 0)
00030     {
00031         perror("pthread_attr_setschedpolicy");
00032         throw "Error";
00033     }
00034
00035     // Change priority for the thread to mPriority
00036     struct sched_param param;
00037     if ( (ret = pthread_attr_getschedparam(&mAttr, &param)) != 0)
00038     {
00039         perror("pthread_attr_getschedparam");
00040         throw "Error";
00041     }
00042     param.__sched_priority = mPriority;
00043
00044     if ( (ret = pthread_attr_setschedparam(&mAttr, &param)) != 0)
00045     {
00046         perror("pthread_attr_setschedparam");
00047         throw "Error";
00048     }
00049
00050     /*
00051     * Set inherit scheduler attribut to PTHREAD_EXPLICIT_SCHED
00052     * otherwise the schedule attributes in mAttr will be ignored
00053     * and the same settings as the main thread will be inherited.
00054     */
00055     if ( (ret = pthread_attr_setinheritsched(&mAttr, PTHREAD_EXPLICIT_SCHED))
!= 0)
00056     {
00057         perror("pthread_attr_setinheritsched ");
00058         throw "Error";
00059     }
00060 }
00061
00062 RtThread::~RtThread()
00063 {
00064     /*
00065     * Make sure that the thread terminated properly
00066     * before deleting the instance
00067     */
00068     this->join();
00069     int ret;
00070     if ( (ret = pthread_attr_destroy(&mAttr)) != 0)
00071     {
00072         perror("pthread_attr_destroy");
00073         throw "Error";
00074     }
00075 }
00076
00077 inline
00078 pthread_t RtThread::getThreadId() const
00079 {
00080     return mId;
00081 }
00082
00083 inline
00084 int RtThread::getPriority() const
00085 {
00086     return mPriority;
00087 }
00088
00089 void RtThread::start(void *arg)
00090 {
00091     int ret;
00092     mArgs = arg;
00093     /*
00094     * Since pthread_create is a C library function, the 3rd argument is
00095     * a global function that will be executed by the thread. In C++, we
00096     * emulate the global function using the static member function that
00097     * is called exec. The 4th argument is the actual argument passed to

```



```

00098      * the function exec. Here we use this pointer, which is an instance
00099      * of the Thread class.
00100      */
00101
00102      if ( (ret = pthread_create(&mId, &mAttr, &RtThread::exec, this)) !=0)
00103      {
00104          perror("thread_create ");
00105          throw "Error";
00106      }
00107      mStarted = true;
00108  }
00109
00110  bool RtThread::join(int timeout_ms)
00111  {
00112      //Allow the thread to wait for the termination status
00113      if (mStarted)
00114      {
00115          if(timeout_ms == 0)
00116          {
00117              if (pthread_join(mId, NULL) != 0) return false;
00118          }
00119          else
00120          {
00121              struct timespec ts;
00122              if (clock_gettime(CLOCK_REALTIME, &ts) == -1)
00123              {
00124                  perror("clock_gettime ");
00125                  throw "Error";
00126              }
00127              ts.tv_sec  += timeout_ms / 1000;
00128              ts.tv_nsec += timeout_ms * 1000000;
00129
00130              int result = pthread_timedjoin_np(mId, NULL, &ts);
00131              if (result == ETIMEDOUT)
00132                  return false;
00133          }
00134          return true;
00135      }
00136
00137      return false;
00138  }
00139
00140  // Function which is actually executed by the thread
00141  void * RtThread::exec(void *thr)
00142  {
00143      reinterpret_cast<RtThread *> (thr)->run();
00144      return NULL;
00145  }
00146
00147

```