DarkRoom

Release 0.0.

May 04, 2017

Usage and Installation

The DarkRoom project provides 3D tracking of arbitrary objects. It is based on the HTC vive lighthouse system, though except for the two lighthouses, the project is completely independent from proprietary content. We documented our project on hackaday aswell: https://hackaday.io/project/19570-htc-vive-lighthouse-custom-tracking

CHAPTER 1

Relevant Background Information and Pre-Requisits

software: A user should be familiar with linux and the ROS build procedure.

A developer should have experience in c++ and be familiar with ROS and arduino. For understanding the fpga code, a developer needs to get familiar with verilog and vhdl.

hardware:

A user will probably receive a ready system, consisting of:

- two HTC lighthouses
- a tracked object

A developer will need the following components:

- two HTC lighthouses
- one altera de0-nano fpga
- one breakout board and a couple of wires and resistors
- one esp8266
- one imu MPU6050
- one-many custom made lighthouse sensors

Contents:

Dependencies (on Ubuntu 16.04)

ROS kinetic: `sudo apt install ros-kinetic-desktop-full`

eigen3: `sudo apt install libeigen3-dev`

yaml-cpp: `sudo apt install libyaml-cpp-dev`

opencv 3: `sudo apt install libopencv-dev`

google protobuf: `sudo apt install protobuf-compiler protobuf-c-compiler`

qt5

Installation

```
cd DarkRoom
catkin_make
source devel/setup.bash
```

Getting started

The GUI software is a rviz plugin. Start rviz and add the DarkRoom panel under Add Panel. You will also need to edit fixed frame to 'world' and add a Marker and tf visualization.

Turn on your tracked object now.

tick "listen for new objects" in rviz DarkRoom panel. After a few seconds your tracked object should be listed in the Panel with its respective IP.

Context

Todo

Create a black box view of your software within its intendend environment. Identify all neighboring systems and specify all logical business data that is exchanged with the system under development.

List of all (a-l-l) neighboring systems.

Motivation.

Understanding the information exchange with neighboring systems (i.e. all input flows and all output flows).

Conventions

We follow the coding guidelines:

Todo

Define what coding guideline they should use, if you differ from the ones below.

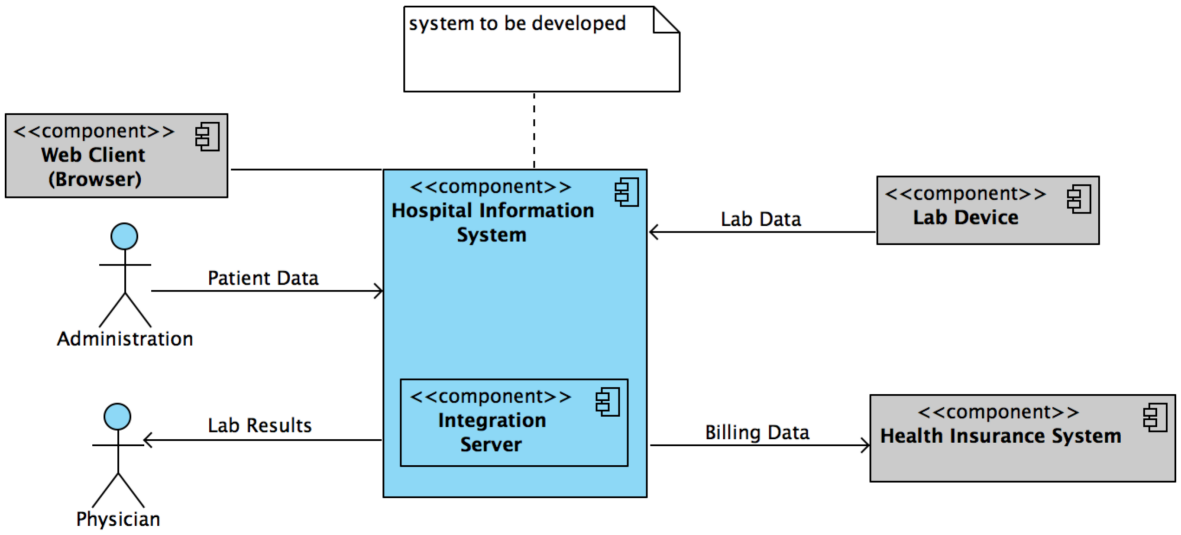

Fig. 1.1: UML System Context

UML-type context diagram - shows the birds eye view of the system (black box) described by this architecture within the ecosystem it is to be placed in. Shows orbit level interfaces on the user interaction and component scope.

Table 1.1: Coding Guidelines

Lan-guage	Guideline	Tools
Python	https://www.python.org/dev/peps/pep-0008/	
C++	http://wiki.ros.org/CppStyleGuide	clang-format: https://github.com/davetcoleman/roscpp_code_format

Architecture Constraints

Todo

Describe what constraints someone further developing this software should adhere to, and why. Should they not use tool x or operating system y... You can use the table as below, or just put a list.

Technical Constraints / Runtime Interface Requirements

Todo

List all technical constraints in this section. This category covers runtime interface requirements and constraints such as:

- Hard- and software infrastructure
- Applied technologies - Operating systems - Middleware - Databases - Programming languages

Table 1.2: Hardware Constraints

Constraint Name	Description
Altera FPGA	All code is highly specific to the Altera FPGA. Intel has bought Altera and aims to integrate their SoC with FPGAs. We are on the right horse!
Intel RealSense	Only the intel real sense can sense it..

Table 1.3: Software Constraints

Constraint Name	Description
Altera FPGA	All code is highly specific to the Altera FPGA. Intel has bought Altera and aims to integrate their SoC with FPGAs. We are on the right horse!
Intel RealSense	Only the intel real sense can sense it...

Table 1.4: Operating System Constraints

Constraint Name	Description
Windows 8 or higher	Due to the Intel RealSense SDK only being supported on Windows, we are stuck with Windows

Table 1.5: Programming Constraints

Constraint Name	Description
CouchDB	We have to use the CouchDB because the type of data we have to store changes at runtime...

Technical Interfaces

This section describes the data interfaces to other systems around it. It follows 3 of the levels of interoperability (**IO**):

Todo

For all your interfaces, define their first 3 levels of interoperability. You can use your doxygen documented source code to i.e. show all members of a class. Find more on the Breathe Documentation

- Technical interoperability - Datastreams btwn systems. i.e. TCP/IP, RS232, ...
- Syntactic interoperability - Units within the stream. i.e. XML, CSV, HL7, DICOM
- Semantic interoperability - Common definition of unit meaning.

Powerlink for MotorControl

Technical IO

Powerlink

Syntactic IO

SDO, PDO, NMT messages

Semantic IO

The package names or registers or

Warning: doxygenclass: Cannot find class “Nutshell” in doxygen xml output for project “DarkRoom” from directory: doxyxml/

User Interfaces

How does a user interact with the system.

Todo

If you have a user interacting with the finished system, and especially if you have a UI or GUI, describe how it can be used. A good way of doing this, is by building a

- state transition diagram aka the ‘Dynamic UI Behaviuour’ and
- a mockup/picture of every screen the user can see - aka ‘Static UI’

Don’t overdo it...

Dynamic UI Behaviour

Static UI

Design Decisions

This can document decisions on the design of the software.

Todo

If you want to use it, keep track of decisions that someone further developing this software or using it in the future might want to know about. This might save them time or clarify expectations. You can put them as a list, or whatever.

Things to consider:

- What exactly is the challenge?
- Why is it relevant for the architecture?
- What consequences does the decision have?
- Which constraints do you have to keep in mind?
- What factors influence the decision?
- Which assumption have you made?

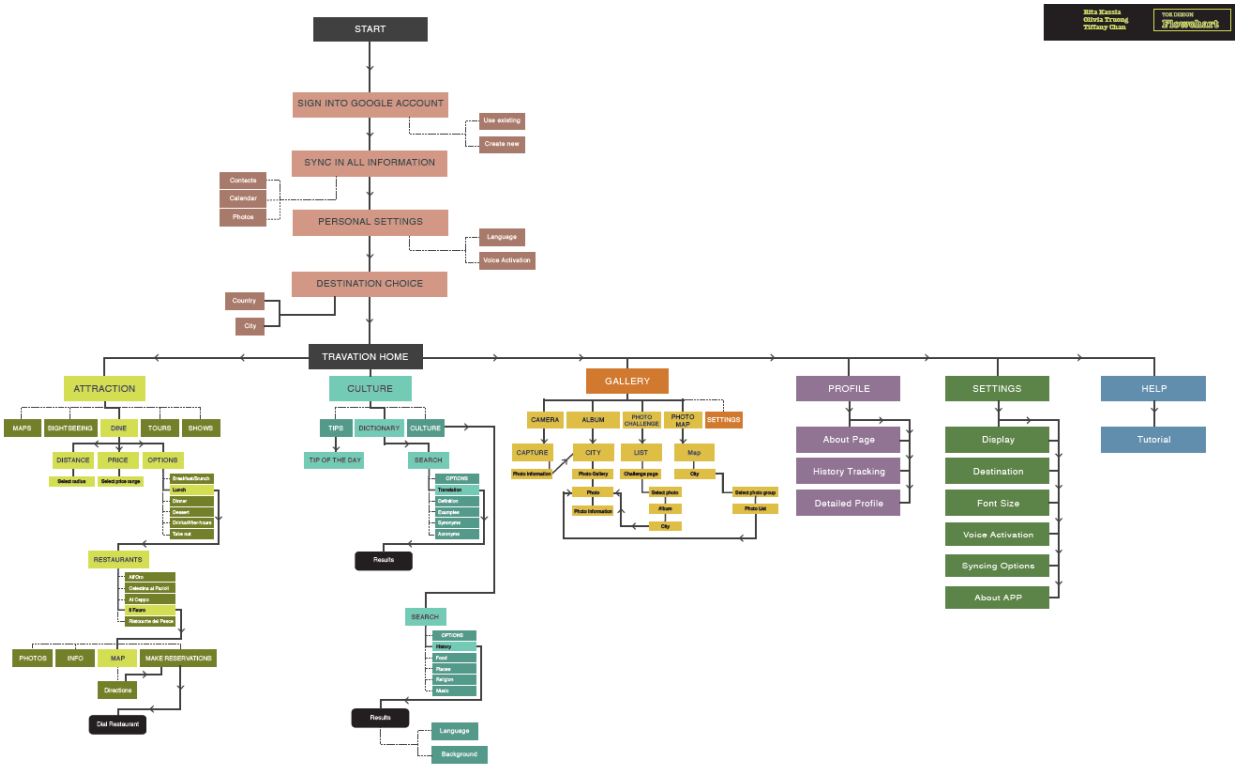

Fig. 1.2: Diagram showing the dynamic behaviour of the user interface i.e. State diagram

- How can you check those assumptions?
- Which risks are you facing?
- Which alternative options did you consider?
- How do you judge each one?
- Which alternatives are you excluding deliberately?
- Who (if not you) has decided?
- How has the decision been justified?
- When did you decide?

Public Interfaces

struct `_DarkRoomProtobuf_commandObject`

Public Members

int32_t **`command`**

struct `_DarkRoomProtobuf_configObject`

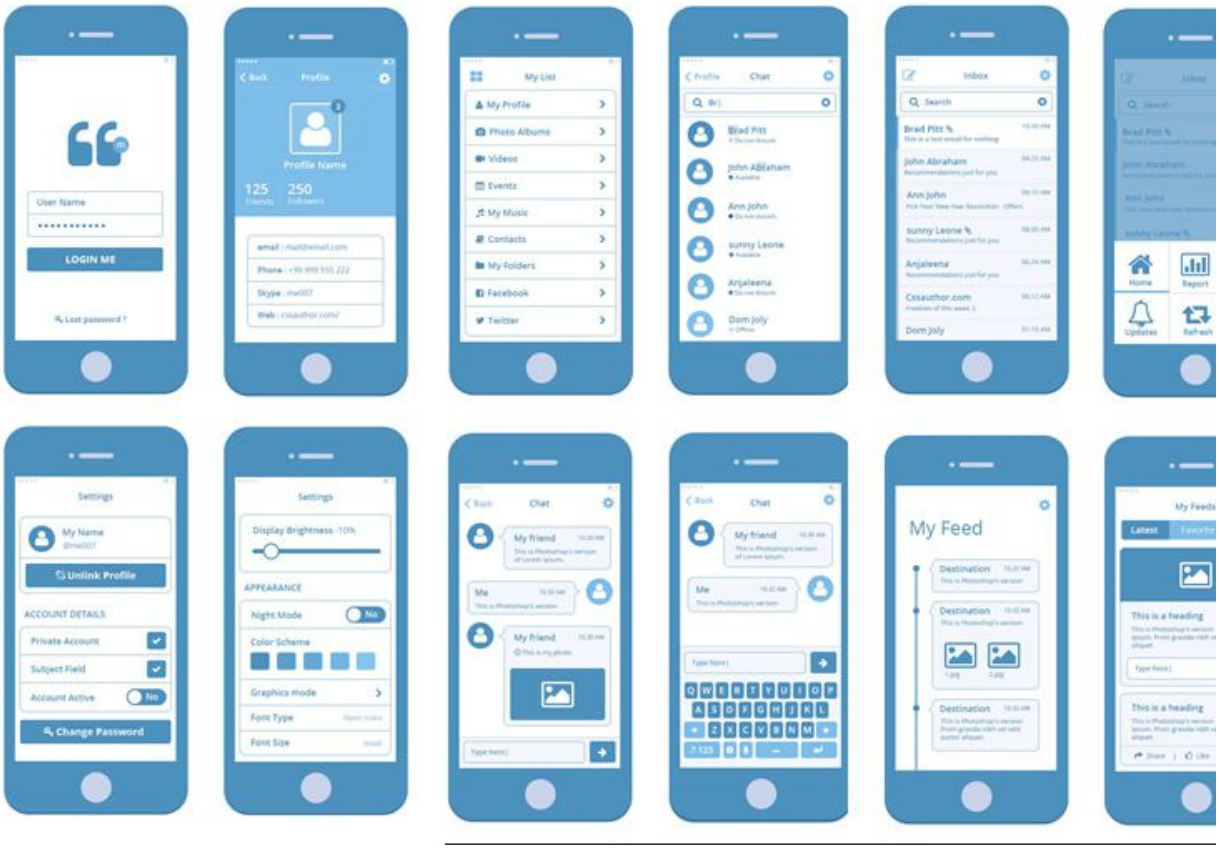

Fig. 1.3: Mock-up screens of the individual views/screens of the GUI i.e. Wireframes, whiteboard sketches

Public Members

uint32_t **ip**

int32_t **logging_port**

int32_t **sensor_port**

int32_t **imu_port**

struct _DarkRoomProtobuf_imuObject

Public Members

int32_t **id**

pb_size_t **quaternion_count**

float **quaternion**[4]

pb_size_t **acc_count**

int32_t **acc**[3]

pb_size_t **gravity_count**

float **gravity**[3]

struct _DarkRoomProtobuf_loggingObject

Public Members

bool **has_message**

char **message**[40]

struct _DarkRoomProtobuf_trackedObjectConfig

Public Members

uint32_t **ip**

int32_t **command_port**

struct _FIFO128sweep

Public Members

uint8_t **mRead**

uint8_t **mWrite**

Sweep ***mBuffer**[32]

struct _FIFO128t

Public Members

uint8_t **mRead**

uint8_t **mWrite**

uint32_t **mBuffer**[32]

struct COLOR

Public Functions

COLOR(float *r*, float *g*, float *b*, float *a*)

Public Members

float **r**

float **g**

float **b**

float **a**

class DarkRoom
Inherits from Panel

Public Functions

DarkRoom(QWidget **parent* = 0)

~DarkRoom()

void **load**(**const** rviz::Config &*config*)
Load all configuration data for this panel from the given Config object.

Parameters

- `config`: rviz config file

void **save**(rviz::Config *config*) **const**
Save all configuration data from this panel to the given Config object.

It is important here that you call *save()* on the parent class so the class id and panel name get saved.

Parameters

- `config`: rviz config file

Public Slots

void **connectTo**()
Connects to sensor_port and IP given via GUI.

void **clearAll**()
Clears all visualization markers.

void **resetLighthousePoses**()
Resets the lighthouse poses to inital values.

void **record**()
Toggles recording sensor values for all tracked objects.

void **track**()
Toggles tracking of all objects (ie triangulation)

void **showRays**()
Toggles visualization of lighthouse rays.

void **calibrate**()
Toggles calibration.

void **estimateDistance**()
Toggles distance estimatio for both lighthouses.

void **estimatePose**()

void **pingThread**()

void **switch_lighthouses**()

void **startObjectListener**()

void **rebootESP**()

void **toggleMPU6050**()

Private Functions

void **transformPublisher**()
Is regularily publishing the tf frames (world_vive, lighthouse1, lighthouse2)

void **objectListener**()
Ping thread regularily listening for new objects.

bool **checkIfObjectAlreadyExists**(**const** string &*IP*)
This functions looks through trackedObjects ListWidget if the object is already being tracked.

Return true (object already exists), false (new object)

Parameters

- `IP`: IP of new object

bool **checkIfObjectAlreadyExists**(**const** uint32_t &*IP*)

Private Members

boost::shared_ptr<*std*::thread> **transform_thread** = nullptr

boost::shared_ptr<*std*::thread> **object_listener_thread**

ros::NodeHandlePtr **nh**

boost::shared_ptr<ros::AsyncSpinner> **spinner**

ros::Publisher **visualization_pub**

tf::TransformListener **tf_listener**

tf::TransformBroadcaster **tf_broadcaster**

tf::Transform **lighthouse1**

tf::Transform **lighthouse2**

tf::Transform **tf_world**

bool **lighthouse_switch** = true

bool **publish_transform** = true

bool **add_new_objects** = true

int **message_counter** = 0

int **object_counter** = 0

map<int, *TrackedObjectPtr*> **trackedObjects**

map<int, string> **IPs**

string **server_IP**

int **server_port**

UDPSocketPtr **ping_socket**

mutex **m_lockMutex**

template <typename *_Scalar*, int *NX* = Dynamic, int *NY* = Dynamic>
struct **Functor**

Public Types

enum **[anonymous]**

Values:

InputsAtCompileTime = NX

ValuesAtCompileTime = NY

typedef _Scalar **Scalar**

typedef Matrix<*Scalar*, *InputsAtCompileTime*, 1> **InputType**

typedef Matrix<*Scalar*, *ValuesAtCompileTime*, 1> **ValueType**

typedef Matrix<*Scalar*, *ValuesAtCompileTime*, *InputsAtCompileTime*> **JacobianType**

Public Functions

Functor()

Functor(int *inputs*, int *values*)

int **inputs**() **const**

int **values**() **const**

Public Members

const int **m_inputs**

const int **m_values**

class I2Cdev

Public Functions

I2Cdev()
Default constructor.

Public Static Functions

int8_t **readBit**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitNum*, uint8_t *data, uint16_t *timeout* = I2Cdev::readTimeout)
Read a single bit from an 8-bit device register.

Return Status of read operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to read from
- `bitNum`: Bit position to read (0-7)
- `data`: Container for single bit value
- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

int8_t **readBitW**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitNum*, uint16_t *data, uint16_t *timeout* = I2Cdev::readTimeout)
Read a single bit from a 16-bit device register.

Return Status of read operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to read from
- `bitNum`: Bit position to read (0-15)

- `data`: Container for single bit value
- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

int8_t **readBits**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitStart*, uint8_t *length*, uint8_t **data*, uint16_t *timeout* = I2Cdev::readTimeout)

Read multiple bits from an 8-bit device register.

Return Status of read operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to read from
- `bitStart`: First bit position to read (0-7)
- `length`: Number of bits to read (not more than 8)
- `data`: Container for right-aligned value (i.e. '101' read from any bitStart position will equal 0x05)
- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

int8_t **readBitsW**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitStart*, uint8_t *length*, uint16_t **data*, uint16_t *timeout* = I2Cdev::readTimeout)

Read multiple bits from a 16-bit device register.

Return Status of read operation (1 = success, 0 = failure, -1 = timeout)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to read from
- `bitStart`: First bit position to read (0-15)
- `length`: Number of bits to read (not more than 16)
- `data`: Container for right-aligned value (i.e. '101' read from any bitStart position will equal 0x05)
- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

int8_t **readByte**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t **data*, uint16_t *timeout* = I2Cdev::readTimeout)

Read single byte from an 8-bit device register.

Return Status of read operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to read from
- `data`: Container for byte value read from device

- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

int8_t **readWord**(uint8_t *devAddr*, uint8_t *regAddr*, uint16_t **data*, uint16_t *timeout* = I2Cdev::readTimeout)
Read single word from a 16-bit device register.

Return Status of read operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to read from
- `data`: Container for word value read from device
- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

int8_t **readBytes**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *length*, uint8_t **data*, uint16_t *timeout* = I2Cdev::readTimeout)
Read multiple bytes from an 8-bit device register.

Return Number of bytes read (-1 indicates failure)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: First register regAddr to read from
- `length`: Number of bytes to read
- `data`: Buffer to store read data in
- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

int8_t **readWords**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *length*, uint16_t **data*, uint16_t *timeout* = I2Cdev::readTimeout)
Read multiple words from a 16-bit device register.

Return Number of words read (-1 indicates failure)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: First register regAddr to read from
- `length`: Number of words to read
- `data`: Buffer to store read data in
- `timeout`: Optional read timeout in milliseconds (0 to disable, leave off to use default class value in *I2Cdev::readTimeout*)

bool **writeBit**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitNum*, uint8_t *data*)
write a single bit in an 8-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to write to
- `bitNum`: Bit position to write (0-7)
- `value`: New bit value to write

bool **`writeBitW`**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitNum*, uint16_t *data*)
write a single bit in a 16-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to write to
- `bitNum`: Bit position to write (0-15)
- `value`: New bit value to write

bool **`writeBits`**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitStart*, uint8_t *length*, uint8_t *data*)
Write multiple bits in an 8-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to write to
- `bitStart`: First bit position to write (0-7)
- `length`: Number of bits to write (not more than 8)
- `data`: Right-aligned value to write

bool **`writeBitsW`**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *bitStart*, uint8_t *length*, uint16_t *data*)
Write multiple bits in a 16-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register regAddr to write to
- `bitStart`: First bit position to write (0-15)
- `length`: Number of bits to write (not more than 16)
- `data`: Right-aligned value to write

bool **`writeByte`**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *data*)
Write single byte to an 8-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register address to write to
- `data`: New byte value to write

bool **`writeWord`**(uint8_t *devAddr*, uint8_t *regAddr*, uint16_t *data*)
Write single word to a 16-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: Register address to write to
- `data`: New word value to write

bool **`writeBytes`**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *length*, uint8_t **data*)
Write multiple bytes to an 8-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: First register address to write to
- `length`: Number of bytes to write
- `data`: Buffer to copy new data from

bool **`writeWords`**(uint8_t *devAddr*, uint8_t *regAddr*, uint8_t *length*, uint16_t **data*)
Write multiple words to a 16-bit device register.

Return Status of operation (true = success)

Parameters

- `devAddr`: I2C slave device address
- `regAddr`: First register address to write to
- `length`: Number of words to write
- `data`: Buffer to copy new data from

Public Static Attributes

uint16_t **`readTimeout`** = I2CDEV_DEFAULT_READ_TIMEOUT
Default timeout value for read operations.

Set this to 0 to disable timeout detection.

struct `LogI`

Public Members

void (***const** **GetLogString**) (*TLogLevel*, **const** char *)

void (***const** **MakefLogString**) (*TLogLevel*, **const** char *, float)

void (***const** **MakeldLogString**) (*TLogLevel*, **const** char *, long int)

void (***const** **MakedLogString**) (*TLogLevel*, **const** char *, int)

TLogLevel (***const** **GetReportingLevel**) (void)

void (***const** **setFlushInterface**) (*FlushInterface*)

void (***const** **setLoggingLevel**) (*TLogLevel*)

struct **LogString**

Public Members

size_t **length**

char ***buff**

TLogLevel **messageLevel**

FlushI **flushI**

class **MPU6050**

Public Functions

MPU6050()
Default constructor, uses default I2C address.

See MPU6050_DEFAULT_ADDRESS

MPU6050(uint8_t *address*)
Specific address constructor.

See MPU6050_DEFAULT_ADDRESS

See MPU6050_ADDRESS_AD0_LOW

See MPU6050_ADDRESS_AD0_HIGH

Parameters

- `address`: I2C address

void **initialize**()
Power on and prepare for general usage.

This will activate the device and take it out of sleep mode (which must be done after start-up). This function also sets both the accelerometer and the gyroscope to their most sensitive settings, namely +/- 2g and +/- 250 degrees/sec, and sets the clock source to use the X Gyro for reference, which is slightly better than the default internal clock source.

bool **testConnection**()
Verify the I2C connection.

Make sure the device is connected and responds as expected.

Return True if connection is valid, false otherwise

uint8_t **getAuxVDDIOLevel**()
Get the auxiliary I2C supply voltage level.

When set to 1, the auxiliary I2C bus high logic level is VDD. When cleared to 0, the auxiliary I2C bus high logic level is VLOGIC. This does not apply to the MPU-6000, which does not have a VLOGIC pin.

Return I2C supply voltage level (0=VLOGIC, 1=VDD)

void **setAuxVDDIOLevel**(uint8_t *level*)
Set the auxiliary I2C supply voltage level.

When set to 1, the auxiliary I2C bus high logic level is VDD. When cleared to 0, the auxiliary I2C bus high logic level is VLOGIC. This does not apply to the MPU-6000, which does not have a VLOGIC pin.

Parameters

- `level`: I2C supply voltage level (0=VLOGIC, 1=VDD)

uint8_t **getRate**()
Get gyroscope output rate divider.

The sensor register output, FIFO output, DMP sampling, Motion detection, Zero Motion detection, and Free Fall detection are all based on the Sample Rate. The Sample Rate is generated by dividing the gyroscope output rate by SMPLRT_DIV:

Sample Rate = Gyroscope Output Rate / (1 + SMPLRT_DIV)

where Gyroscope Output Rate = 8kHz when the DLPF is disabled (DLPF_CFG = 0 or 7), and 1kHz when the DLPF is enabled (see Register 26).

Note: The accelerometer output rate is 1kHz. This means that for a Sample Rate greater than 1kHz, the same accelerometer sample may be output to the FIFO, DMP, and sensor registers more than once.

For a diagram of the gyroscope and accelerometer signal paths, see Section 8 of the MPU-6000/MPU-6050 Product Specification document.

Return Current sample rate

See MPU6050_RA_SMPLRT_DIV

void **setRate**(uint8_t *rate*)
Set gyroscope sample rate divider.

See *getRate()*

See MPU6050_RA_SMPLRT_DIV

Parameters

- `rate`: New sample rate divider

uint8_t **getExternalFrameSync**()
Get external FSYNC configuration.

Configures the external Frame Synchronization (FSYNC) pin sampling. An external signal connected to the FSYNC pin can be sampled by configuring EXT_SYNC_SET. Signal changes to the FSYNC pin are

latched so that short strobes may be captured. The latched FSYNC signal will be sampled at the Sampling Rate, as defined in register 25. After sampling, the latch will reset to the current FSYNC signal state.

The sampled value will be reported in place of the least significant bit in a sensor data register determined by the value of EXT_SYNC_SET according to the following table.

Return FSYNC configuration value

void **setExternalFrameSync**(uint8_t *sync*)
Set external FSYNC configuration.

See *getExternalFrameSync()*

See MPU6050_RA_CONFIG

Parameters

- `sync`: New FSYNC configuration value

uint8_t **getDLPFMode**()
Get digital low-pass filter configuration.

The DLPF_CFG parameter sets the digital low pass filter configuration. It also determines the internal sampling rate used by the device as shown in the table below.

Note: The accelerometer output rate is 1kHz. This means that for a Sample Rate greater than 1kHz, the same accelerometer sample may be output to the FIFO, DMP, and sensor registers more than once.

Return DLFP configuration

See MPU6050_RA_CONFIG

See MPU6050_CFG_DLPF_CFG_BIT

See MPU6050_CFG_DLPF_CFG_LENGTH

void **setDLPFMode**(uint8_t *bandwidth*)
Set digital low-pass filter configuration.

See getDLPFBandwidth()

See MPU6050_DLPF_BW_256

See MPU6050_RA_CONFIG

See MPU6050_CFG_DLPF_CFG_BIT

See MPU6050_CFG_DLPF_CFG_LENGTH

Parameters

- `mode`: New DLFP configuration setting

uint8_t **getFullScaleGyroRange**()
Get full-scale gyroscope range.

The FS_SEL parameter allows setting the full-scale range of the gyro sensors, as described in the table below.

Return Current full-scale gyroscope range setting

See MPU6050_GYRO_FS_250

See MPU6050_RA_GYRO_CONFIG

See MPU6050_GCONFIG_FS_SEL_BIT

See MPU6050_GCONFIG_FS_SEL_LENGTH

void **setFullScaleGyroRange**(uint8_t *range*)
Set full-scale gyroscope range.

See getFullScaleRange()

See MPU6050_GYRO_FS_250

See MPU6050_RA_GYRO_CONFIG

See MPU6050_GCONFIG_FS_SEL_BIT

See MPU6050_GCONFIG_FS_SEL_LENGTH

Parameters

- `range`: New full-scale gyroscope range value

uint8_t **getAccelXSelfTestFactoryTrim**()
Get self-test factory trim value for accelerometer X axis.

Return factory trim value

See MPU6050_RA_SELF_TEST_X

uint8_t **getAccelYSelfTestFactoryTrim**()
Get self-test factory trim value for accelerometer Y axis.

Return factory trim value

See MPU6050_RA_SELF_TEST_Y

uint8_t **getAccelZSelfTestFactoryTrim**()
Get self-test factory trim value for accelerometer Z axis.

Return factory trim value

See MPU6050_RA_SELF_TEST_Z

uint8_t **getGyroXSelfTestFactoryTrim**()
Get self-test factory trim value for gyro X axis.

Return factory trim value

See MPU6050_RA_SELF_TEST_X

uint8_t **getGyroYSelfTestFactoryTrim**()
Get self-test factory trim value for gyro Y axis.

Return factory trim value

See MPU6050_RA_SELF_TEST_Y

uint8_t **getGyroZSelfTestFactoryTrim**()
Get self-test factory trim value for gyro Z axis.

Return factory trim value

See MPU6050_RA_SELF_TEST_Z

bool **getAccelXSelfTest**()
Get self-test enabled setting for accelerometer X axis.

Return Self-test enabled value

See MPU6050_RA_ACCEL_CONFIG

void **setAccelXSelfTest**(bool *enabled*)
Get self-test enabled setting for accelerometer X axis.

See MPU6050_RA_ACCEL_CONFIG

Parameters

- `enabled`: Self-test enabled value

bool **getAccelYSelfTest**()
Get self-test enabled value for accelerometer Y axis.

Return Self-test enabled value

See MPU6050_RA_ACCEL_CONFIG

void **setAccelYSelfTest**(bool *enabled*)
Get self-test enabled value for accelerometer Y axis.

See MPU6050_RA_ACCEL_CONFIG

Parameters

- `enabled`: Self-test enabled value

bool **getAccelZSelfTest**()
Get self-test enabled value for accelerometer Z axis.

Return Self-test enabled value

See MPU6050_RA_ACCEL_CONFIG

void **setAccelZSelfTest**(bool *enabled*)
Set self-test enabled value for accelerometer Z axis.

See MPU6050_RA_ACCEL_CONFIG

Parameters

- `enabled`: Self-test enabled value

uint8_t **getFullScaleAccelRange**()
Get full-scale accelerometer range.

The FS_SEL parameter allows setting the full-scale range of the accelerometer sensors, as described in the table below.

Return Current full-scale accelerometer range setting

See MPU6050_ACCEL_FS_2

See MPU6050_RA_ACCEL_CONFIG

See MPU6050_ACONFIG_AFS_SEL_BIT

See MPU6050_ACONFIG_AFS_SEL_LENGTH

void **setFullScaleAccelRange**(uint8_t *range*)
Set full-scale accelerometer range.

See *getFullScaleAccelRange()*

Parameters

- `range`: New full-scale accelerometer range setting

uint8_t **getDHPFMode**()
Get the high-pass filter configuration.

The DHPF is a filter module in the path leading to motion detectors (Free Fall, Motion threshold, and Zero Motion). The high pass filter output is not available to the data registers (see Figure in Section 8 of the MPU-6000/ MPU-6050 Product Specification document).

The high pass filter has three modes:

Return Current high-pass filter configuration

See MPU6050_DHPF_RESET

See MPU6050_RA_ACCEL_CONFIG

void **setDHPFMode**(uint8_t *mode*)
Set the high-pass filter configuration.

See *setDHPFMode()*

See MPU6050_DHPF_RESET

See MPU6050_RA_ACCEL_CONFIG

Parameters

- `bandwidth`: New high-pass filter configuration

uint8_t **getFreefallDetectionThreshold**()
Get free-fall event acceleration threshold.

This register configures the detection threshold for Free Fall event detection. The unit of FF_THR is 1LSB = 2mg. Free Fall is detected when the absolute value of the accelerometer measurements for the three axes are each less than the detection threshold. This condition increments the Free Fall duration counter (Register 30). The Free Fall interrupt is triggered when the Free Fall duration counter reaches the time specified in FF_DUR.

For more details on the Free Fall detection interrupt, see Section 8.2 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Return Current free-fall acceleration threshold value (LSB = 2mg)

See MPU6050_RA_FF_THR

void **setFreefallDetectionThreshold**(uint8_t *threshold*)
Get free-fall event acceleration threshold.

See *getFreefallDetectionThreshold()*

See MPU6050_RA_FF_THR

Parameters

- `threshold`: New free-fall acceleration threshold value (LSB = 2mg)

uint8_t **getFreefallDetectionDuration**()
Get free-fall event duration threshold.

This register configures the duration counter threshold for Free Fall event detection. The duration counter ticks at 1kHz, therefore FF_DUR has a unit of 1 LSB = 1 ms.

The Free Fall duration counter increments while the absolute value of the accelerometer measurements are each less than the detection threshold (Register 29). The Free Fall interrupt is triggered when the Free Fall duration counter reaches the time specified in this register.

For more details on the Free Fall detection interrupt, see Section 8.2 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Return Current free-fall duration threshold value (LSB = 1ms)

See MPU6050_RA_FF_DUR

void **setFreefallDetectionDuration**(uint8_t *duration*)
Get free-fall event duration threshold.

See *getFreefallDetectionDuration()*

See MPU6050_RA_FF_DUR

Parameters

- `duration`: New free-fall duration threshold value (LSB = 1ms)

uint8_t **getMotionDetectionThreshold**()
Get motion detection event acceleration threshold.

This register configures the detection threshold for Motion interrupt generation. The unit of MOT_THR is 1LSB = 2mg. Motion is detected when the absolute value of any of the accelerometer measurements exceeds this Motion detection threshold. This condition increments the Motion detection duration counter (Register 32). The Motion detection interrupt is triggered when the Motion Detection counter reaches the time count specified in MOT_DUR (Register 32).

The Motion interrupt will indicate the axis and polarity of detected motion in MOT_DETECT_STATUS (Register 97).

For more details on the Motion detection interrupt, see Section 8.3 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Return Current motion detection acceleration threshold value (LSB = 2mg)

See MPU6050_RA_MOT_THR

void **setMotionDetectionThreshold**(uint8_t *threshold*)
Set motion detection event acceleration threshold.

See *getMotionDetectionThreshold()*

See MPU6050_RA_MOT_THR

Parameters

- `threshold`: New motion detection acceleration threshold value (LSB = 2mg)

uint8_t **getMotionDetectionDuration**()
Get motion detection event duration threshold.

This register configures the duration counter threshold for Motion interrupt generation. The duration counter ticks at 1 kHz, therefore MOT_DUR has a unit of 1LSB = 1ms. The Motion detection duration counter increments when the absolute value of any of the accelerometer measurements exceeds the Motion detection threshold (Register 31). The Motion detection interrupt is triggered when the Motion detection counter reaches the time count specified in this register.

For more details on the Motion detection interrupt, see Section 8.3 of the MPU-6000/MPU-6050 Product Specification document.

Return Current motion detection duration threshold value (LSB = 1ms)

See MPU6050_RA_MOT_DUR

void **setMotionDetectionDuration**(uint8_t *duration*)
Set motion detection event duration threshold.

See *getMotionDetectionDuration()*

See MPU6050_RA_MOT_DUR

Parameters

- `duration`: New motion detection duration threshold value (LSB = 1ms)

uint8_t **getZeroMotionDetectionThreshold**()
Get zero motion detection event acceleration threshold.

This register configures the detection threshold for Zero Motion interrupt generation. The unit of ZRMOT_THR is 1LSB = 2mg. Zero Motion is detected when the absolute value of the accelerometer measurements for the 3 axes are each less than the detection threshold. This condition increments the Zero Motion duration counter (Register 34). The Zero Motion interrupt is triggered when the Zero Motion duration counter reaches the time count specified in ZRMOT_DUR (Register 34).

Unlike Free Fall or Motion detection, Zero Motion detection triggers an interrupt both when Zero Motion is first detected and when Zero Motion is no longer detected.

When a zero motion event is detected, a Zero Motion Status will be indicated in the MOT_DETECT_STATUS register (Register 97). When a motion-to-zero-motion condition is detected, the status bit is set to 1. When a zero-motion-to- motion condition is detected, the status bit is set to 0.

For more details on the Zero Motion detection interrupt, see Section 8.4 of the MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and 58 of this document.

Return Current zero motion detection acceleration threshold value (LSB = 2mg)

See MPU6050_RA_ZRMOT_THR

void **setZeroMotionDetectionThreshold**(uint8_t *threshold*)
Set zero motion detection event acceleration threshold.

See *getZeroMotionDetectionThreshold()*

See MPU6050_RA_ZRMOT_THR

Parameters

- `threshold`: New zero motion detection acceleration threshold value (LSB = 2mg)

uint8_t **getZeroMotionDetectionDuration**()
Get zero motion detection event duration threshold.

This register configures the duration counter threshold for Zero Motion interrupt generation. The duration counter ticks at 16 Hz, therefore ZRMOT_DUR has a unit of 1 LSB = 64 ms. The Zero Motion duration counter increments while the absolute value of the accelerometer measurements are each less than the detection threshold (Register 33). The Zero Motion interrupt is triggered when the Zero Motion duration counter reaches the time count specified in this register.

For more details on the Zero Motion detection interrupt, see Section 8.4 of the MPU-6000/MPU-6050 Product Specification document, as well as Registers 56 and 58 of this document.

Return Current zero motion detection duration threshold value (LSB = 64ms)

See MPU6050_RA_ZRMOT_DUR

void **setZeroMotionDetectionDuration**(uint8_t *duration*)
Set zero motion detection event duration threshold.

See *getZeroMotionDetectionDuration()*

See MPU6050_RA_ZRMOT_DUR

Parameters

- `duration`: New zero motion detection duration threshold value (LSB = 1ms)

bool **getTempFIFOEnabled**()
Get temperature FIFO enabled value.

When set to 1, this bit enables TEMP_OUT_H and TEMP_OUT_L (Registers 65 and 66) to be written into the FIFO buffer.

Return Current temperature FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setTempFIFOEnabled**(bool *enabled*)
Set temperature FIFO enabled value.

See *getTempFIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New temperature FIFO enabled value

bool **getXGyroFIFOEnabled**()
Get gyroscope X-axis FIFO enabled value.

When set to 1, this bit enables GYRO_XOUT_H and GYRO_XOUT_L (Registers 67 and 68) to be written into the FIFO buffer.

Return Current gyroscope X-axis FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setXGyroFIFOEnabled**(bool *enabled*)
Set gyroscope X-axis FIFO enabled value.

See *getXGyroFIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New gyroscope X-axis FIFO enabled value

bool **getYGyroFIFOEnabled**()
Get gyroscope Y-axis FIFO enabled value.

When set to 1, this bit enables GYRO_YOUT_H and GYRO_YOUT_L (Registers 69 and 70) to be written into the FIFO buffer.

Return Current gyroscope Y-axis FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setYGyroFIFOEnabled**(bool *enabled*)
Set gyroscope Y-axis FIFO enabled value.

See *getYGyroFIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New gyroscope Y-axis FIFO enabled value

bool **getZGyroFIFOEnabled**()
Get gyroscope Z-axis FIFO enabled value.

When set to 1, this bit enables GYRO_ZOUT_H and GYRO_ZOUT_L (Registers 71 and 72) to be written into the FIFO buffer.

Return Current gyroscope Z-axis FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setZGyroFIFOEnabled**(bool *enabled*)
Set gyroscope Z-axis FIFO enabled value.

See *getZGyroFIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New gyroscope Z-axis FIFO enabled value

bool **getAccelFIFOEnabled**()
Get accelerometer FIFO enabled value.

When set to 1, this bit enables ACCEL_XOUT_H, ACCEL_XOUT_L, ACCEL_YOUT_H, ACCEL_YOUT_L, ACCEL_ZOUT_H, and ACCEL_ZOUT_L (Registers 59 to 64) to be written into the FIFO buffer.

Return Current accelerometer FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setAccelFIFOEnabled**(bool *enabled*)
Set accelerometer FIFO enabled value.

See *getAccelFIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New accelerometer FIFO enabled value

bool **getSlave2FIFOEnabled**()
Get Slave 2 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 2 to be written into the FIFO buffer.

Return Current Slave 2 FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setSlave2FIFOEnabled**(bool *enabled*)
Set Slave 2 FIFO enabled value.

See *getSlave2FIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New Slave 2 FIFO enabled value

bool **getSlave1FIFOEnabled**()
Get Slave 1 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 1 to be written into the FIFO buffer.

Return Current Slave 1 FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setSlave1FIFOEnabled**(bool *enabled*)
Set Slave 1 FIFO enabled value.

See *getSlave1FIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New Slave 1 FIFO enabled value

bool **getSlave0FIFOEnabled**()
Get Slave 0 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 0 to be written into the FIFO buffer.

Return Current Slave 0 FIFO enabled value

See MPU6050_RA_FIFO_EN

void **setSlave0FIFOEnabled**(bool *enabled*)
Set Slave 0 FIFO enabled value.

See *getSlave0FIFOEnabled()*

See MPU6050_RA_FIFO_EN

Parameters

- `enabled`: New Slave 0 FIFO enabled value

bool **getMultiMasterEnabled**()
Get multi-master enabled value.

Multi-master capability allows multiple I2C masters to operate on the same bus. In circuits where multi-master capability is required, set MULT_MST_EN to 1. This will increase current drawn by approximately 30uA.

In circuits where multi-master capability is required, the state of the I2C bus must always be monitored by each separate I2C Master. Before an I2C Master can assume arbitration of the bus, it must first confirm that no other I2C Master has arbitration of the bus. When MULT_MST_EN is set to 1, the MPU-60X0's bus arbitration detection logic is turned on, enabling it to detect when the bus is available.

Return Current multi-master enabled value

See MPU6050_RA_I2C_MST_CTRL

void **setMultiMasterEnabled**(bool *enabled*)
Set multi-master enabled value.

See *getMultiMasterEnabled()*

See MPU6050_RA_I2C_MST_CTRL

Parameters

- `enabled`: New multi-master enabled value

bool **getWaitForExternalSensorEnabled**()
Get wait-for-external-sensor-data enabled value.

When the WAIT_FOR_ES bit is set to 1, the Data Ready interrupt will be delayed until External *Sensor* data from the Slave Devices are loaded into the EXT_SENS_DATA registers. This is used to ensure that both the internal sensor data (i.e. from gyro and accel) and external sensor data have been loaded to their respective data registers (i.e. the data is synced) when the Data Ready interrupt is triggered.

Return Current wait-for-external-sensor-data enabled value

See MPU6050_RA_I2C_MST_CTRL

void **setWaitForExternalSensorEnabled**(bool *enabled*)
Set wait-for-external-sensor-data enabled value.

See *getWaitForExternalSensorEnabled()*

See MPU6050_RA_I2C_MST_CTRL

Parameters

- `enabled`: New wait-for-external-sensor-data enabled value

bool **getSlave3FIFOEnabled**()
Get Slave 3 FIFO enabled value.

When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96) associated with Slave 3 to be written into the FIFO buffer.

Return Current Slave 3 FIFO enabled value

See MPU6050_RA_MST_CTRL

void **setSlave3FIFOEnabled**(bool *enabled*)
Set Slave 3 FIFO enabled value.

See *getSlave3FIFOEnabled()*

See MPU6050_RA_MST_CTRL

Parameters

- `enabled`: New Slave 3 FIFO enabled value

bool **getSlaveReadWriteTransitionEnabled**()
Get slave read/write transition enabled value.

The I2C_MST_P_NSR bit configures the I2C Master's transition from one slave read to the next slave read. If the bit equals 0, there will be a restart between reads. If the bit equals 1, there will be a stop followed by a start of the following read. When a write transaction follows a read transaction, the stop followed by a start of the successive write will be always used.

Return Current slave read/write transition enabled value

See MPU6050_RA_I2C_MST_CTRL

void **setSlaveReadWriteTransitionEnabled**(bool *enabled*)
Set slave read/write transition enabled value.

See *getSlaveReadWriteTransitionEnabled()*

See MPU6050_RA_I2C_MST_CTRL

Parameters

- `enabled`: New slave read/write transition enabled value

uint8_t **getMasterClockSpeed**()
Get I2C master clock speed.

I2C_MST_CLK is a 4 bit unsigned value which configures a divider on the MPU-60X0 internal 8MHz clock. It sets the I2C master clock speed according to the following table:

Return Current I2C master clock speed

See MPU6050_RA_I2C_MST_CTRL

void **setMasterClockSpeed**(uint8_t *speed*)
Set I2C master clock speed.

speed Current I2C master clock speed

See MPU6050_RA_I2C_MST_CTRL

uint8_t **getSlaveAddress**(uint8_t *num*)
Get the I2C address of the specified slave (0-3).

Note that Bit 7 (MSB) controls read/write mode. If Bit 7 is set, it's a read operation, and if it is cleared, then it's a write operation. The remaining bits (6-0) are the 7-bit device address of the slave device.

In read mode, the result of the read is placed in the lowest available EXT_SENS_DATA register. For further information regarding the allocation of read results, please refer to the EXT_SENS_DATA register description (Registers 73 - 96).

The MPU-6050 supports a total of five slaves, but Slave 4 has unique characteristics, and so it has its own functions (getSlave4* and setSlave4*).

I2C data transactions are performed at the Sample Rate, as defined in Register 25. The user is responsible for ensuring that I2C data transactions to and from each enabled Slave can be completed within a single period of the Sample Rate.

The I2C slave access rate can be reduced relative to the Sample Rate. This reduced access rate is determined by I2C_MST_DLY (Register 52). Whether a slave's access rate is reduced relative to the Sample Rate is determined by I2C_MST_DELAY_CTRL (Register 103).

The processing order for the slaves is fixed. The sequence followed for processing the slaves is Slave 0, Slave 1, Slave 2, Slave 3 and Slave 4. If a particular Slave is disabled it will be skipped.

Each slave can either be accessed at the sample rate or at a reduced sample rate. In a case where some slaves are accessed at the Sample Rate and some slaves are accessed at the reduced rate, the sequence of accessing the slaves (Slave 0 to Slave 4) is still followed. However, the reduced rate slaves will be skipped if their access rate dictates that they should not be accessed during that particular cycle. For further information regarding the reduced access rate, please refer to Register 52. Whether a slave is accessed at the Sample Rate or at the reduced rate is determined by the Delay Enable bits in Register 103.

Return Current address for specified slave

See MPU6050_RA_I2C_SLV0_ADDR

Parameters

- `num`: Slave number (0-3)

void **setSlaveAddress**(uint8_t *num*, uint8_t *address*)
Set the I2C address of the specified slave (0-3).

See *getSlaveAddress()*

See MPU6050_RA_I2C_SLV0_ADDR

Parameters

- `num`: Slave number (0-3)
- `address`: New address for specified slave

uint8_t **getSlaveRegister**(uint8_t *num*)
Get the active internal register for the specified slave (0-3).

Read/write operations for this slave will be done to whatever internal register address is stored in this MPU register.

The MPU-6050 supports a total of five slaves, but Slave 4 has unique characteristics, and so it has its own functions.

Return Current active register for specified slave

See MPU6050_RA_I2C_SLV0_REG

Parameters

- `num`: Slave number (0-3)

void **setSlaveRegister**(uint8_t *num*, uint8_t *reg*)
Set the active internal register for the specified slave (0-3).

See *getSlaveRegister()*

See MPU6050_RA_I2C_SLV0_REG

Parameters

- `num`: Slave number (0-3)
- `reg`: New active register for specified slave

bool **getSlaveEnabled**(uint8_t *num*)
Get the enabled value for the specified slave (0-3).

When set to 1, this bit enables Slave 0 for data transfer operations. When cleared to 0, this bit disables Slave 0 from data transfer operations.

Return Current enabled value for specified slave

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)

void **setSlaveEnabled**(uint8_t *num*, bool *enabled*)
Set the enabled value for the specified slave (0-3).

See *getSlaveEnabled()*

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)
- `enabled`: New enabled value for specified slave

bool **getSlaveWordByteSwap**(uint8_t *num*)
Get word pair byte-swapping enabled for the specified slave (0-3).

When set to 1, this bit enables byte swapping. When byte swapping is enabled, the high and low bytes of a word pair are swapped. Please refer to I2C_SLV0_GRP for the pairing convention of the word pairs. When cleared to 0, bytes transferred to and from Slave 0 will be written to EXT_SENS_DATA registers in the order they were transferred.

Return Current word pair byte-swapping enabled value for specified slave

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)

void **setSlaveWordByteSwap**(uint8_t *num*, bool *enabled*)
Set word pair byte-swapping enabled for the specified slave (0-3).

See *getSlaveWordByteSwap()*

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)
- `enabled`: New word pair byte-swapping enabled value for specified slave

bool **getSlaveWriteMode**(uint8_t *num*)
Get write mode for the specified slave (0-3).

When set to 1, the transaction will read or write data only. When cleared to 0, the transaction will write a register address prior to reading or writing data. This should equal 0 when specifying the register address within the Slave device to/from which the ensuing data transaction will take place.

Return Current write mode for specified slave (0 = register address + data, 1 = data only)

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)

void **setSlaveWriteMode**(uint8_t *num*, bool *mode*)
Set write mode for the specified slave (0-3).

See *getSlaveWriteMode()*

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)
- `mode`: New write mode for specified slave (0 = register address + data, 1 = data only)

bool **getSlaveWordGroupOffset**(uint8_t *num*)
Get word pair grouping order offset for the specified slave (0-3).

This sets specifies the grouping order of word pairs received from registers. When cleared to 0, bytes from register addresses 0 and 1, 2 and 3, etc (even, then odd register addresses) are paired to form a word. When

set to 1, bytes from register addresses are paired 1 and 2, 3 and 4, etc. (odd, then even register addresses) are paired to form a word.

Return Current word pair grouping order offset for specified slave

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)

void **setSlaveWordGroupOffset** (uint8_t *num*, bool *enabled*)
Set word pair grouping order offset for the specified slave (0-3).

See *getSlaveWordGroupOffset()*

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)
- `enabled`: New word pair grouping order offset for specified slave

uint8_t **getSlaveDataLength** (uint8_t *num*)
Get number of bytes to read for the specified slave (0-3).

Specifies the number of bytes transferred to and from Slave 0. Clearing this bit to 0 is equivalent to disabling the register by writing 0 to I2C_SLV0_EN.

Return Number of bytes to read for specified slave

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)

void **setSlaveDataLength** (uint8_t *num*, uint8_t *length*)
Set number of bytes to read for the specified slave (0-3).

See *getSlaveDataLength()*

See MPU6050_RA_I2C_SLV0_CTRL

Parameters

- `num`: Slave number (0-3)
- `length`: Number of bytes to read for specified slave

uint8_t **getSlave4Address** ()
Get the I2C address of Slave 4.

Note that Bit 7 (MSB) controls read/write mode. If Bit 7 is set, it's a read operation, and if it is cleared, then it's a write operation. The remaining bits (6-0) are the 7-bit device address of the slave device.

Return Current address for Slave 4

See *getSlaveAddress()*

See MPU6050_RA_I2C_SLV4_ADDR

void **setSlave4Address**(uint8_t *address*)
Set the I2C address of Slave 4.

See *getSlave4Address()*

See MPU6050_RA_I2C_SLV4_ADDR

Parameters

- `address`: New address for Slave 4

uint8_t **getSlave4Register**()
Get the active internal register for the Slave 4.

Read/write operations for this slave will be done to whatever internal register address is stored in this MPU register.

Return Current active register for Slave 4

See MPU6050_RA_I2C_SLV4_REG

void **setSlave4Register**(uint8_t *reg*)
Set the active internal register for Slave 4.

See *getSlave4Register()*

See MPU6050_RA_I2C_SLV4_REG

Parameters

- `reg`: New active register for Slave 4

void **setSlave4OutputByte**(uint8_t *data*)
Set new byte to write to Slave 4.

This register stores the data to be written into the Slave 4. If I2C_SLV4_RW is set 1 (set to read), this register has no effect.

See MPU6050_RA_I2C_SLV4_DO

Parameters

- `data`: New byte to write to Slave 4

bool **getSlave4Enabled**()
Get the enabled value for the Slave 4.

When set to 1, this bit enables Slave 4 for data transfer operations. When cleared to 0, this bit disables Slave 4 from data transfer operations.

Return Current enabled value for Slave 4

See MPU6050_RA_I2C_SLV4_CTRL

void **setSlave4Enabled**(bool *enabled*)
Set the enabled value for Slave 4.

See *getSlave4Enabled()*

See MPU6050_RA_I2C_SLV4_CTRL

Parameters

- `enabled`: New enabled value for Slave 4

bool **getSlave4InterruptEnabled**()
Get the enabled value for Slave 4 transaction interrupts.

When set to 1, this bit enables the generation of an interrupt signal upon completion of a Slave 4 transaction. When cleared to 0, this bit disables the generation of an interrupt signal upon completion of a Slave 4 transaction. The interrupt status can be observed in Register 54.

Return Current enabled value for Slave 4 transaction interrupts.

See MPU6050_RA_I2C_SLV4_CTRL

void **setSlave4InterruptEnabled**(bool *enabled*)
Set the enabled value for Slave 4 transaction interrupts.

See *getSlave4InterruptEnabled()*

See MPU6050_RA_I2C_SLV4_CTRL

Parameters

- `enabled`: New enabled value for Slave 4 transaction interrupts.

bool **getSlave4WriteMode**()
Get write mode for Slave 4.

When set to 1, the transaction will read or write data only. When cleared to 0, the transaction will write a register address prior to reading or writing data. This should equal 0 when specifying the register address within the Slave device to/from which the ensuing data transaction will take place.

Return Current write mode for Slave 4 (0 = register address + data, 1 = data only)

See MPU6050_RA_I2C_SLV4_CTRL

void **setSlave4WriteMode**(bool *mode*)
Set write mode for the Slave 4.

See *getSlave4WriteMode()*

See MPU6050_RA_I2C_SLV4_CTRL

Parameters

- `mode`: New write mode for Slave 4 (0 = register address + data, 1 = data only)

uint8_t **getSlave4MasterDelay**()
Get Slave 4 master delay value.

This configures the reduced access rate of I2C slaves relative to the Sample Rate. When a slave's access rate is decreased relative to the Sample Rate, the slave is accessed every:

```
1 / (1 + I2C_MST_DLY) samples
```

This base Sample Rate in turn is determined by SMPLRT_DIV (register 25) and DLPF_CFG (register 26). Whether a slave's access rate is reduced relative to the Sample Rate is determined by I2C_MST_DELAY_CTRL (register 103). For further information regarding the Sample Rate, please refer to register 25.

Return Current Slave 4 master delay value

See MPU6050_RA_I2C_SLV4_CTRL

void **setSlave4MasterDelay**(uint8_t *delay*)
Set Slave 4 master delay value.

See *getSlave4MasterDelay()*

See MPU6050_RA_I2C_SLV4_CTRL

Parameters

- `delay`: New Slave 4 master delay value

uint8_t **getSlate4InputByte**()
Get last available byte read from Slave 4.

This register stores the data read from Slave 4. This field is populated after a read transaction.

Return Last available byte read from to Slave 4

See MPU6050_RA_I2C_SLV4_DI

bool **getPassthroughStatus**()
Get FSYNC interrupt status.

This bit reflects the status of the FSYNC interrupt from an external device into the MPU-60X0. This is used as a way to pass an external interrupt through the MPU-60X0 to the host application processor. When set to 1, this bit will cause an interrupt if FSYNC_INT_EN is asserted in INT_PIN_CFG (Register 55).

Return FSYNC interrupt status

See MPU6050_RA_I2C_MST_STATUS

bool **getSlave4IsDone**()
Get Slave 4 transaction done status.

Automatically sets to 1 when a Slave 4 transaction has completed. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted and if the SLV_4_DONE_INT bit is asserted in the I2C_SLV4_CTRL register (Register 52).

Return Slave 4 transaction done status

See MPU6050_RA_I2C_MST_STATUS

bool **getLostArbitration**()
Get master arbitration lost status.

This bit automatically sets to 1 when the I2C Master has lost arbitration of the auxiliary I2C bus (an error condition). This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Return Master arbitration lost status

See MPU6050_RA_I2C_MST_STATUS

bool **getSlave4Nack**()
Get Slave 4 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 4. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Return Slave 4 NACK interrupt status

See MPU6050_RA_I2C_MST_STATUS

bool **getSlave3Nack**()
Get Slave 3 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 3. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Return Slave 3 NACK interrupt status

See MPU6050_RA_I2C_MST_STATUS

bool **getSlave2Nack**()
Get Slave 2 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 2. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Return Slave 2 NACK interrupt status

See MPU6050_RA_I2C_MST_STATUS

bool **getSlave1Nack**()
Get Slave 1 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 1. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Return Slave 1 NACK interrupt status

See MPU6050_RA_I2C_MST_STATUS

bool **getSlave0Nack**()
Get Slave 0 NACK status.

This bit automatically sets to 1 when the I2C Master receives a NACK in a transaction with Slave 0. This triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.

Return Slave 0 NACK interrupt status

See MPU6050_RA_I2C_MST_STATUS

bool **getInterruptMode**()
Get interrupt logic level mode.

Will be set 0 for active-high, 1 for active-low.

Return Current interrupt mode (0=active-high, 1=active-low)

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_INT_LEVEL_BIT

void **setInterruptMode**(bool *mode*)
Set interrupt logic level mode.

See *getInterruptMode()*

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_INT_LEVEL_BIT

Parameters

- `mode`: New interrupt mode (0=active-high, 1=active-low)

bool **getInterruptDrive**()
Get interrupt drive mode.

Will be set 0 for push-pull, 1 for open-drain.

Return Current interrupt drive mode (0=push-pull, 1=open-drain)

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_INT_OPEN_BIT

void **setInterruptDrive**(bool *drive*)
Set interrupt drive mode.

See *getInterruptDrive()*

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_INT_OPEN_BIT

Parameters

- `drive`: New interrupt drive mode (0=push-pull, 1=open-drain)

bool **getInterruptLatch**()
Get interrupt latch mode.

Will be set 0 for 50us-pulse, 1 for latch-until-int-cleared.

Return Current latch mode (0=50us-pulse, 1=latch-until-int-cleared)

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_LATCH_INT_EN_BIT

void **setInterruptLatch**(bool *latch*)
Set interrupt latch mode.

See *getInterruptLatch()*

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_LATCH_INT_EN_BIT

Parameters

- `latch`: New latch mode (0=50us-pulse, 1=latch-until-int-cleared)

bool **getInterruptLatchClear**()
Get interrupt latch clear mode.

Will be set 0 for status-read-only, 1 for any-register-read.

Return Current latch clear mode (0=status-read-only, 1=any-register-read)

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_INT_RD_CLEAR_BIT

void **setInterruptLatchClear**(bool *clear*)
Set interrupt latch clear mode.

See *getInterruptLatchClear()*

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_INT_RD_CLEAR_BIT

Parameters

- `clear`: New latch clear mode (0=status-read-only, 1=any-register-read)

bool **getFSyncInterruptLevel**()
Get FSYNC interrupt logic level mode.

Return Current FSYNC interrupt mode (0=active-high, 1=active-low)

See getFSyncInterruptMode()

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT

void **setFSyncInterruptLevel**(bool *level*)
Set FSYNC interrupt logic level mode.

See getFSyncInterruptMode()

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT

Parameters

- `mode`: New FSYNC interrupt mode (0=active-high, 1=active-low)

bool **getFSyncInterruptEnabled**()
Get FSYNC pin interrupt enabled setting.

Will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled setting

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_FSYNC_INT_EN_BIT

void **setFSyncInterruptEnabled**(bool *enabled*)
Set FSYNC pin interrupt enabled setting.

See *getFSyncInterruptEnabled()*

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_FSYNC_INT_EN_BIT

Parameters

- `enabled`: New FSYNC pin interrupt enabled setting

bool **getI2CBypassEnabled**()
Get I2C bypass enabled status.

When this bit is equal to 1 and I2C_MST_EN (Register 106 bit[5]) is equal to 0, the host application processor will be able to directly access the auxiliary I2C bus of the MPU-60X0. When this bit is equal to 0, the host application processor will not be able to directly access the auxiliary I2C bus of the MPU-60X0 regardless of the state of I2C_MST_EN (Register 106 bit[5]).

Return Current I2C bypass enabled status

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_I2C_BYPASS_EN_BIT

void **setI2CBypassEnabled**(bool *enabled*)
Set I2C bypass enabled status.

When this bit is equal to 1 and I2C_MST_EN (Register 106 bit[5]) is equal to 0, the host application processor will be able to directly access the auxiliary I2C bus of the MPU-60X0. When this bit is equal to 0, the host application processor will not be able to directly access the auxiliary I2C bus of the MPU-60X0 regardless of the state of I2C_MST_EN (Register 106 bit[5]).

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_I2C_BYPASS_EN_BIT

Parameters

- `enabled`: New I2C bypass enabled status

bool **getClockOutputEnabled**()
Get reference clock output enabled status.

When this bit is equal to 1, a reference clock output is provided at the CLKOUT pin. When this bit is equal to 0, the clock output is disabled. For further information regarding CLKOUT, please refer to the MPU-60X0 Product Specification document.

Return Current reference clock output enabled status

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_CLKOUT_EN_BIT

void **setClockOutputEnabled**(bool *enabled*)
Set reference clock output enabled status.

When this bit is equal to 1, a reference clock output is provided at the CLKOUT pin. When this bit is equal to 0, the clock output is disabled. For further information regarding CLKOUT, please refer to the MPU-60X0 Product Specification document.

See MPU6050_RA_INT_PIN_CFG

See MPU6050_INTCFG_CLKOUT_EN_BIT

Parameters

- `enabled`: New reference clock output enabled status

uint8_t **getIntEnabled**()
Get full interrupt enabled status.

Full register byte for all interrupts, for quick reading. Each bit will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled status

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_FF_BIT

void **setIntEnabled**(uint8_t *enabled*)
Set full interrupt enabled status.

Full register byte for all interrupts, for quick reading. Each bit should be set 0 for disabled, 1 for enabled.

See *getIntFreefallEnabled()*

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_FF_BIT

Parameters

- `enabled`: New interrupt enabled status

bool **getIntFreefallEnabled**()
Get Free Fall interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled status

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_FF_BIT

void **setIntFreefallEnabled**(bool *enabled*)
Set Free Fall interrupt enabled status.

See *getIntFreefallEnabled()*

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_FF_BIT

Parameters

- `enabled`: New interrupt enabled status

bool **getIntMotionEnabled**()
Get Motion Detection interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled status

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_MOT_BIT

void **setIntMotionEnabled**(bool *enabled*)
Set Motion Detection interrupt enabled status.

See *getIntMotionEnabled()*

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_MOT_BIT

Parameters

- `enabled`: New interrupt enabled status

bool **getIntZeroMotionEnabled**()
Get Zero Motion Detection interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled status

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_ZMOT_BIT

void **setIntZeroMotionEnabled**(bool *enabled*)
Set Zero Motion Detection interrupt enabled status.

See *getIntZeroMotionEnabled()*

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_ZMOT_BIT

Parameters

- `enabled`: New interrupt enabled status

bool **getIntFIFOBufferOverflowEnabled**()
Get FIFO Buffer Overflow interrupt enabled status.

Will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled status

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_FIFO_OFLOW_BIT

void **setIntFIFOBufferOverflowEnabled**(bool *enabled*)
Set FIFO Buffer Overflow interrupt enabled status.

See *getIntFIFOBufferOverflowEnabled()*

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_FIFO_OFLOW_BIT

Parameters

- `enabled`: New interrupt enabled status

bool **getIntI2CMasterEnabled**()
Get I2C Master interrupt enabled status.

This enables any of the I2C Master interrupt sources to generate an interrupt. Will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled status

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_I2C_MST_INT_BIT

void **setIntI2CMasterEnabled**(bool *enabled*)
Set I2C Master interrupt enabled status.

See *getIntI2CMasterEnabled()*

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_I2C_MST_INT_BIT

Parameters

- `enabled`: New interrupt enabled status

bool **getIntDataReadyEnabled**()
Get Data Ready interrupt enabled setting.

This event occurs each time a write operation to all of the sensor registers has been completed. Will be set 0 for disabled, 1 for enabled.

Return Current interrupt enabled status

See MPU6050_RA_INT_ENABLE

See MPU6050_INTERRUPT_DATA_RDY_BIT

void **setIntDataReadyEnabled**(bool *enabled*)
Set Data Ready interrupt enabled status.

See *getIntDataReadyEnabled()*

See MPU6050_RA_INT_CFG

See MPU6050_INTERRUPT_DATA_RDY_BIT

Parameters

- `enabled`: New interrupt enabled status

uint8_t **getIntStatus**()
Get full set of interrupt status bits.

These bits clear to 0 after the register has been read. Very useful for getting multiple INT statuses, since each single bit read clears all of them because it has to read the whole byte.

Return Current interrupt status

See MPU6050_RA_INT_STATUS

bool **getIntFreefallStatus**()
Get Free Fall interrupt status.

This bit automatically sets to 1 when a Free Fall interrupt has been generated. The bit clears to 0 after the register has been read.

Return Current interrupt status

See MPU6050_RA_INT_STATUS

See MPU6050_INTERRUPT_FF_BIT

bool **getIntMotionStatus**()
Get Motion Detection interrupt status.

This bit automatically sets to 1 when a Motion Detection interrupt has been generated. The bit clears to 0 after the register has been read.

Return Current interrupt status

See MPU6050_RA_INT_STATUS

See MPU6050_INTERRUPT_MOT_BIT

bool **getIntZeroMotionStatus**()
Get Zero Motion Detection interrupt status.

This bit automatically sets to 1 when a Zero Motion Detection interrupt has been generated. The bit clears to 0 after the register has been read.

Return Current interrupt status

See MPU6050_RA_INT_STATUS

See MPU6050_INTERRUPT_ZMOT_BIT

bool **getIntFIFOBufferOverflowStatus**()
Get FIFO Buffer Overflow interrupt status.

This bit automatically sets to 1 when a Free Fall interrupt has been generated. The bit clears to 0 after the register has been read.

Return Current interrupt status

See MPU6050_RA_INT_STATUS

See MPU6050_INTERRUPT_FIFO_OFLOW_BIT

bool **getIntI2CMasterStatus**()
Get I2C Master interrupt status.

This bit automatically sets to 1 when an I2C Master interrupt has been generated. For a list of I2C Master interrupts, please refer to Register 54. The bit clears to 0 after the register has been read.

Return Current interrupt status

See MPU6050_RA_INT_STATUS

See MPU6050_INTERRUPT_I2C_MST_INT_BIT

bool **getIntDataReadyStatus**()
Get Data Ready interrupt status.

This bit automatically sets to 1 when a Data Ready interrupt has been generated. The bit clears to 0 after the register has been read.

Return Current interrupt status

See MPU6050_RA_INT_STATUS

See MPU6050_INTERRUPT_DATA_RDY_BIT

void **getMotion9**(int16_t *ax, int16_t *ay, int16_t *az, int16_t *gx, int16_t *gy, int16_t *gz, int16_t *mx, int16_t *my, int16_t *mz)
Get raw 9-axis motion sensor readings (accel/gyro/compass).

FUNCTION NOT FULLY IMPLEMENTED YET.

See *getMotion6()*

See *getAcceleration()*

See *getRotation()*

See MPU6050_RA_ACCEL_XOUT_H

Parameters

- `ax`: 16-bit signed integer container for accelerometer X-axis value
- `ay`: 16-bit signed integer container for accelerometer Y-axis value
- `az`: 16-bit signed integer container for accelerometer Z-axis value
- `gx`: 16-bit signed integer container for gyroscope X-axis value
- `gy`: 16-bit signed integer container for gyroscope Y-axis value

- gz: 16-bit signed integer container for gyroscope Z-axis value
- mx: 16-bit signed integer container for magnetometer X-axis value
- my: 16-bit signed integer container for magnetometer Y-axis value
- mz: 16-bit signed integer container for magnetometer Z-axis value

void **getMotion6**(int16_t *ax, int16_t *ay, int16_t *az, int16_t *gx, int16_t *gy, int16_t *gz)
Get raw 6-axis motion sensor readings (accel/gyro).

Retrieves all currently available motion sensor values.

See *getAcceleration()*

See *getRotation()*

See MPU6050_RA_ACCEL_XOUT_H

Parameters

- ax: 16-bit signed integer container for accelerometer X-axis value
- ay: 16-bit signed integer container for accelerometer Y-axis value
- az: 16-bit signed integer container for accelerometer Z-axis value
- gx: 16-bit signed integer container for gyroscope X-axis value
- gy: 16-bit signed integer container for gyroscope Y-axis value
- gz: 16-bit signed integer container for gyroscope Z-axis value

void **getAcceleration**(int16_t *x, int16_t *y, int16_t *z)
Get 3-axis accelerometer readings.

These registers store the most recent accelerometer measurements. Accelerometer measurements are written to these registers at the Sample Rate as defined in Register 25.

The accelerometer measurement registers, along with the temperature measurement registers, gyroscope measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the accelerometer sensors' internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Each 16-bit accelerometer measurement has a full scale defined in ACCEL_FS (Register 28). For each full scale setting, the accelerometers' sensitivity per LSB in ACCEL_xOUT is shown in the table below:

See MPU6050_RA_GYRO_XOUT_H

Parameters

- x: 16-bit signed integer container for X-axis acceleration
- y: 16-bit signed integer container for Y-axis acceleration
- z: 16-bit signed integer container for Z-axis acceleration

int16_t **getAccelerationX**()
Get X-axis accelerometer reading.

Return X-axis acceleration measurement in 16-bit 2's complement format

See *getMotion6()*

See MPU6050_RA_ACCEL_XOUT_H

int16_t **getAccelerationY**()
Get Y-axis accelerometer reading.

Return Y-axis acceleration measurement in 16-bit 2's complement format

See *getMotion6()*

See MPU6050_RA_ACCEL_YOUT_H

int16_t **getAccelerationZ**()
Get Z-axis accelerometer reading.

Return Z-axis acceleration measurement in 16-bit 2's complement format

See *getMotion6()*

See MPU6050_RA_ACCEL_ZOUT_H

int16_t **getTemperature**()
Get current internal temperature.

Return Temperature reading in 16-bit 2's complement format

See MPU6050_RA_TEMP_OUT_H

void **getRotation**(int16_t *x, int16_t *y, int16_t *z)
Get 3-axis gyroscope readings.

These gyroscope measurement registers, along with the accelerometer measurement registers, temperature measurement registers, and external sensor data registers, are composed of two sets of registers: an internal register set and a user-facing read register set. The data within the gyroscope sensors' internal register set is always updated at the Sample Rate. Meanwhile, the user-facing read register set duplicates the internal register set's data values whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Each 16-bit gyroscope measurement has a full scale defined in FS_SEL (Register 27). For each full scale setting, the gyroscopes' sensitivity per LSB in GYRO_xOUT is shown in the table below:

See *getMotion6()*

See MPU6050_RA_GYRO_XOUT_H

Parameters

- x: 16-bit signed integer container for X-axis rotation
- y: 16-bit signed integer container for Y-axis rotation
- z: 16-bit signed integer container for Z-axis rotation

int16_t **getRotationX**()
Get X-axis gyroscope reading.

Return X-axis rotation measurement in 16-bit 2's complement format

See *getMotion6()*

See MPU6050_RA_GYRO_XOUT_H

int16_t **getRotationY**()
Get Y-axis gyroscope reading.

Return Y-axis rotation measurement in 16-bit 2's complement format

See *getMotion6()*

See MPU6050_RA_GYRO_YOUT_H

int16_t **getRotationZ**()
Get Z-axis gyroscope reading.

Return Z-axis rotation measurement in 16-bit 2's complement format

See *getMotion6()*

See MPU6050_RA_GYRO_ZOUT_H

uint8_t **getExternalSensorByte**(int *position*)
Read single byte from external sensor data register.

These registers store data read from external sensors by the Slave 0, 1, 2, and 3 on the auxiliary I2C interface. Data read by Slave 4 is stored in I2C_SLV4_DI (Register 53).

External sensor data is written to these registers at the Sample Rate as defined in Register 25. This access rate can be reduced by using the Slave Delay Enable registers (Register 103).

External sensor data registers, along with the gyroscope measurement registers, accelerometer measurement registers, and temperature measurement registers, are composed of two sets of registers: an internal register set and a user-facing read register set.

The data within the external sensors' internal register set is always updated at the Sample Rate (or the reduced access rate) whenever the serial interface is idle. This guarantees that a burst read of sensor registers will read measurements from the same sampling instant. Note that if burst reads are not used, the user is responsible for ensuring a set of single byte reads correspond to a single sampling instant by checking the Data Ready interrupt.

Data is placed in these external sensor data registers according to I2C_SLV0_CTRL, I2C_SLV1_CTRL, I2C_SLV2_CTRL, and I2C_SLV3_CTRL (Registers 39, 42, 45, and 48). When more than zero bytes are read (I2C_SLVx_LEN > 0) from an enabled slave (I2C_SLVx_EN = 1), the slave is read at the Sample Rate (as defined in Register 25) or delayed rate (if specified in Register 52 and 103). During each Sample cycle, slave reads are performed in order of Slave number. If all slaves are enabled with more than zero bytes to be read, the order will be Slave 0, followed by Slave 1, Slave 2, and Slave 3.

Each enabled slave will have EXT_SENS_DATA registers associated with it by number of bytes read (I2C_SLVx_LEN) in order of slave number, starting from EXT_SENS_DATA_00. Note that this means enabling or disabling a slave may change the higher numbered slaves' associated registers. Furthermore, if fewer total bytes are being read from the external sensors as a result of such a change, then the data remaining in the registers which no longer have an associated slave device (i.e. high numbered registers) will remain in these previously allocated registers unless reset.

If the sum of the read lengths of all SLVx transactions exceed the number of available EXT_SENS_DATA registers, the excess bytes will be dropped. There are 24 EXT_SENS_DATA registers and hence the total read lengths between all the slaves cannot be greater than 24 or some bytes will be lost.

Note: Slave 4's behavior is distinct from that of Slaves 0-3. For further information regarding the characteristics of Slave 4, please refer to Registers 49 to 53.

EXAMPLE: Suppose that Slave 0 is enabled with 4 bytes to be read (I2C_SLV0_EN = 1 and I2C_SLV0_LEN = 4) while Slave 1 is enabled with 2 bytes to be read so that I2C_SLV1_EN = 1 and I2C_SLV1_LEN = 2. In such a situation, EXT_SENS_DATA _00 through _03 will be associated with Slave 0, while EXT_SENS_DATA _04 and 05 will be associated with Slave 1. If Slave 2 is enabled as well, registers starting from EXT_SENS_DATA_06 will be allocated to Slave 2.

If Slave 2 is disabled while Slave 3 is enabled in this same situation, then registers starting from EXT_SENS_DATA_06 will be allocated to Slave 3 instead.

REGISTER ALLOCATION FOR DYNAMIC DISABLE VS. NORMAL DISABLE: If a slave is disabled at any time, the space initially allocated to the slave in the EXT_SENS_DATA register, will remain associated with that slave. This is to avoid dynamic adjustment of the register allocation.

The allocation of the EXT_SENS_DATA registers is recomputed only when (1) all slaves are disabled, or (2) the I2C_MST_RST bit is set (Register 106).

This above is also true if one of the slaves gets NACKed and stops functioning.

Return Byte read from register

Parameters

- `position`: Starting position (0-23)

uint16_t **`getExternalSensorWord`**(int *position*)
Read word (2 bytes) from external sensor data registers.

Return Word read from register

See *getExternalSensorByte()*

Parameters

- `position`: Starting position (0-21)

uint32_t **`getExternalSensorDWord`**(int *position*)
Read double word (4 bytes) from external sensor data registers.

Return Double word read from registers

See *getExternalSensorByte()*

Parameters

- `position`: Starting position (0-20)

uint8_t **`getMotionStatus`**()
Get full motion detection status register content (all bits).

Return Motion detection status byte

See MPU6050_RA_MOT_DETECT_STATUS

bool **`getXNegMotionDetected`**()
Get X-axis negative motion detection interrupt status.

Return Motion detection status

See MPU6050_RA_MOT_DETECT_STATUS

See MPU6050_MOTION_MOT_XNEG_BIT

bool **getXPosMotionDetected**()
Get X-axis positive motion detection interrupt status.

Return Motion detection status

See MPU6050_RA_MOT_DETECT_STATUS

See MPU6050_MOTION_MOT_XPOS_BIT

bool **getYNegMotionDetected**()
Get Y-axis negative motion detection interrupt status.

Return Motion detection status

See MPU6050_RA_MOT_DETECT_STATUS

See MPU6050_MOTION_MOT_YNEG_BIT

bool **getYPosMotionDetected**()
Get Y-axis positive motion detection interrupt status.

Return Motion detection status

See MPU6050_RA_MOT_DETECT_STATUS

See MPU6050_MOTION_MOT_YPOS_BIT

bool **getZNegMotionDetected**()
Get Z-axis negative motion detection interrupt status.

Return Motion detection status

See MPU6050_RA_MOT_DETECT_STATUS

See MPU6050_MOTION_MOT_ZNEG_BIT

bool **getZPosMotionDetected**()
Get Z-axis positive motion detection interrupt status.

Return Motion detection status

See MPU6050_RA_MOT_DETECT_STATUS

See MPU6050_MOTION_MOT_ZPOS_BIT

bool **getZeroMotionDetected**()
Get zero motion detection interrupt status.

Return Motion detection status

See MPU6050_RA_MOT_DETECT_STATUS

See MPU6050_MOTION_MOT_ZRMOT_BIT

void **setSlaveOutputByte**(uint8_t *num*, uint8_t *data*)
Write byte to Data Output container for specified slave.

This register holds the output data written into Slave when Slave is set to write mode. For further information regarding Slave control, please refer to Registers 37 to 39 and immediately following.

See MPU6050_RA_I2C_SLV0_DO

Parameters

- `num`: Slave number (0-3)
- `data`: Byte to write

bool **getExternalShadowDelayEnabled**()
Get external data shadow delay enabled status.

This register is used to specify the timing of external sensor data shadowing. When DELAY_ES_SHADOW is set to 1, shadowing of external sensor data is delayed until all data has been received.

Return Current external data shadow delay enabled status.

See MPU6050_RA_I2C_MST_DELAY_CTRL

See MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT

void **setExternalShadowDelayEnabled**(bool *enabled*)
Set external data shadow delay enabled status.

See *getExternalShadowDelayEnabled()*

See MPU6050_RA_I2C_MST_DELAY_CTRL

See MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT

Parameters

- `enabled`: New external data shadow delay enabled status.

bool **getSlaveDelayEnabled**(uint8_t *num*)
Get slave delay enabled status.

When a particular slave delay is enabled, the rate of access for the that slave device is reduced. When a slave's access rate is decreased relative to the Sample Rate, the slave is accessed every:

```
1 / (1 + I2C_MST_DLY) Samples
```

This base Sample Rate in turn is determined by SMPLRT_DIV (register * 25) and DLPF_CFG (register 26).

For further information regarding I2C_MST_DLY, please refer to register 52. For further information regarding the Sample Rate, please refer to register 25.

Return Current slave delay enabled status.

See MPU6050_RA_I2C_MST_DELAY_CTRL

See MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT

Parameters

- `num`: Slave number (0-4)

void **setSlaveDelayEnabled**(uint8_t *num*, bool *enabled*)
Set slave delay enabled status.

See MPU6050_RA_I2C_MST_DELAY_CTRL

See MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT

Parameters

- `num`: Slave number (0-4)
- `enabled`: New slave delay enabled status.

void **resetGyroscopePath**()
Reset gyroscope signal path.

The reset will revert the signal path analog to digital converters and filters to their power up configurations.

See MPU6050_RA_SIGNAL_PATH_RESET

See MPU6050_PATHRESET_GYRO_RESET_BIT

void **resetAccelerometerPath**()
Reset accelerometer signal path.

The reset will revert the signal path analog to digital converters and filters to their power up configurations.

See MPU6050_RA_SIGNAL_PATH_RESET

See MPU6050_PATHRESET_ACCEL_RESET_BIT

void **resetTemperaturePath**()
Reset temperature sensor signal path.

The reset will revert the signal path analog to digital converters and filters to their power up configurations.

See MPU6050_RA_SIGNAL_PATH_RESET

See MPU6050_PATHRESET_TEMP_RESET_BIT

uint8_t **getAccelerometerPowerOnDelay**()
Get accelerometer power-on delay.

The accelerometer data path provides samples to the sensor registers, Motion detection, Zero Motion detection, and Free Fall detection modules. The signal path contains filters which must be flushed on wake-up with new samples before the detection modules begin operations. The default wake-up delay, of 4ms can be lengthened by up to 3ms. This additional delay is specified in ACCEL_ON_DELAY in units of 1 LSB = 1 ms. The user may select any value above zero unless instructed otherwise by InvenSense. Please refer to Section 8 of the MPU-6000/MPU-6050 Product Specification document for further information regarding the detection modules.

Return Current accelerometer power-on delay

See MPU6050_RA_MOT_DETECT_CTRL

See MPU6050_DETECT_ACCEL_ON_DELAY_BIT

void **setAccelerometerPowerOnDelay**(uint8_t *delay*)
Set accelerometer power-on delay.

See *getAccelerometerPowerOnDelay()*

See MPU6050_RA_MOT_DETECT_CTRL

See MPU6050_DETECT_ACCEL_ON_DELAY_BIT

Parameters

- `delay`: New accelerometer power-on delay (0-3)

uint8_t **getFreefallDetectionCounterDecrement**()
Get Free Fall detection counter decrement configuration.

Detection is registered by the Free Fall detection module after accelerometer measurements meet their respective threshold conditions over a specified number of samples. When the threshold conditions are met, the corresponding detection counter increments by 1. The user may control the rate at which the detection counter decrements when the threshold condition is not met by configuring FF_COUNT. The decrement rate can be set according to the following table:

When FF_COUNT is configured to 0 (reset), any non-qualifying sample will reset the counter to 0. For further information on Free Fall detection, please refer to Registers 29 to 32.

Return Current decrement configuration

See MPU6050_RA_MOT_DETECT_CTRL

See MPU6050_DETECT_FF_COUNT_BIT

void **setFreefallDetectionCounterDecrement**(uint8_t *decrement*)
Set Free Fall detection counter decrement configuration.

See *getFreefallDetectionCounterDecrement()*

See MPU6050_RA_MOT_DETECT_CTRL

See MPU6050_DETECT_FF_COUNT_BIT

Parameters

- `decrement`: New decrement configuration value

uint8_t **getMotionDetectionCounterDecrement**()
Get Motion detection counter decrement configuration.

Detection is registered by the Motion detection module after accelerometer measurements meet their respective threshold conditions over a specified number of samples. When the threshold conditions are met, the corresponding detection counter increments by 1. The user may control the rate at which the detection counter decrements when the threshold condition is not met by configuring MOT_COUNT. The decrement rate can be set according to the following table:

When MOT_COUNT is configured to 0 (reset), any non-qualifying sample will reset the counter to 0. For further information on Motion detection, please refer to Registers 29 to 32.

void **setMotionDetectionCounterDecrement**(uint8_t *decrement*)
Set Motion detection counter decrement configuration.

See *getMotionDetectionCounterDecrement()*

See MPU6050_RA_MOT_DETECT_CTRL

See MPU6050_DETECT_MOT_COUNT_BIT

Parameters

- `decrement`: New decrement configuration value

bool **getFIFOEnabled**()
Get FIFO enabled status.

When this bit is set to 0, the FIFO buffer is disabled. The FIFO buffer cannot be written to or read from while disabled. The FIFO buffer's state does not change unless the MPU-60X0 is power cycled.

Return Current FIFO enabled status

See MPU6050_RA_USER_CTRL

See MPU6050_USERCTRL_FIFO_EN_BIT

void **setFIFOEnabled**(bool *enabled*)
Set FIFO enabled status.

See *getFIFOEnabled()*

See MPU6050_RA_USER_CTRL

See MPU6050_USERCTRL_FIFO_EN_BIT

Parameters

- `enabled`: New FIFO enabled status

bool **getI2CMasterModeEnabled**()
Get I2C Master Mode enabled status.

When this mode is enabled, the MPU-60X0 acts as the I2C Master to the external sensor slave devices on the auxiliary I2C bus. When this bit is cleared to 0, the auxiliary I2C bus lines (AUX_DA and AUX_CL) are logically driven by the primary I2C bus (SDA and SCL). This is a precondition to enabling Bypass Mode. For further information regarding Bypass Mode, please refer to Register 55.

Return Current I2C Master Mode enabled status

See MPU6050_RA_USER_CTRL

See MPU6050_USERCTRL_I2C_MST_EN_BIT

void **setI2CMasterModeEnabled**(bool *enabled*)
Set I2C Master Mode enabled status.

See *getI2CMasterModeEnabled()*

See MPU6050_RA_USER_CTRL

See MPU6050_USERCTRL_I2C_MST_EN_BIT

Parameters

- `enabled`: New I2C Master Mode enabled status

void **switchSPIEnabled**(bool *enabled*)
Switch from I2C to SPI mode (MPU-6000 only) If this is set, the primary SPI interface will be enabled in place of the disabled primary I2C interface.

void **resetFIFO**()
Reset the FIFO.

This bit resets the FIFO buffer when set to 1 while FIFO_EN equals 0. This bit automatically clears to 0 after the reset has been triggered.

See MPU6050_RA_USER_CTRL

See MPU6050_USERCTRL_FIFO_RESET_BIT

void **resetI2CMaster**()
Reset the I2C Master.

This bit resets the I2C Master when set to 1 while I2C_MST_EN equals 0. This bit automatically clears to 0 after the reset has been triggered.

See MPU6050_RA_USER_CTRL

See MPU6050_USERCTRL_I2C_MST_RESET_BIT

void **resetSensors**()
Reset all sensor registers and signal paths.

When set to 1, this bit resets the signal paths for all sensors (gyroscopes, accelerometers, and temperature sensor). This operation will also clear the sensor registers. This bit automatically clears to 0 after the reset has been triggered.

When resetting only the signal path (and not the sensor registers), please use Register 104, SIGNAL_PATH_RESET.

See MPU6050_RA_USER_CTRL

See MPU6050_USERCTRL_SIG_COND_RESET_BIT

void **reset**()
Trigger a full device reset.

A small delay of ~50ms may be desirable after triggering a reset.

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_DEVICE_RESET_BIT

bool **getSleepEnabled**()
Get sleep mode status.

Setting the SLEEP bit in the register puts the device into very low power sleep mode. In this mode, only the serial interface and internal registers remain active, allowing for a very low standby current. Clearing this bit puts the device back into normal mode. To save power, the individual standby selections for each of the gyros should be used if any gyro axis is not used by the application.

Return Current sleep mode enabled status

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_SLEEP_BIT

void **setSleepEnabled**(bool *enabled*)
Set sleep mode status.

See *getSleepEnabled()*

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_SLEEP_BIT

Parameters

- `enabled`: New sleep mode enabled status

bool **getWakeCycleEnabled**()
Get wake cycle enabled status.

When this bit is set to 1 and SLEEP is disabled, the MPU-60X0 will cycle between sleep mode and waking up to take a single sample of data from active sensors at a rate determined by LP_WAKE_CTRL (register 108).

Return Current sleep mode enabled status

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_CYCLE_BIT

void **setWakeCycleEnabled**(bool *enabled*)
Set wake cycle enabled status.

See *getWakeCycleEnabled()*

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_CYCLE_BIT

Parameters

- `enabled`: New sleep mode enabled status

bool **getTempSensorEnabled**()
Get temperature sensor enabled status.

Control the usage of the internal temperature sensor.

Note: this register stores the *disabled* value, but for consistency with the rest of the code, the function is named and used with standard true/false values to indicate whether the sensor is enabled or disabled, respectively.

Return Current temperature sensor enabled status

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_TEMP_DIS_BIT

void **setTempSensorEnabled**(bool *enabled*)
Set temperature sensor enabled status.

Note: this register stores the *disabled* value, but for consistency with the rest of the code, the function is named and used with standard true/false values to indicate whether the sensor is enabled or disabled, respectively.

See *getTempSensorEnabled()*

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_TEMP_DIS_BIT

Parameters

- `enabled`: New temperature sensor enabled status

uint8_t **getClockSource**()
Get clock source setting.

Return Current clock source setting

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_CLKSEL_BIT

See MPU6050_PWR1_CLKSEL_LENGTH

void **setClockSource**(uint8_t *source*)
Set clock source setting.

An internal 8MHz oscillator, gyroscope based clock, or external sources can be selected as the MPU-60X0 clock source. When the internal 8 MHz oscillator or an external source is chosen as the clock source, the MPU-60X0 can operate in low power modes with the gyroscopes disabled.

Upon power up, the MPU-60X0 clock source defaults to the internal oscillator. However, it is highly recommended that the device be configured to use one of the gyroscopes (or an external clock source) as the clock reference for improved stability. The clock source can be selected according to the following table:

See *getClockSource()*

See MPU6050_RA_PWR_MGMT_1

See MPU6050_PWR1_CLKSEL_BIT

See MPU6050_PWR1_CLKSEL_LENGTH

Parameters

- `source`: New clock source setting

uint8_t **getWakeFrequency**()
Get wake frequency in Accel-Only Low Power Mode.

The MPU-60X0 can be put into Accerlerometer Only Low Power Mode by setting PWRSEL to 1 in the Power Management 1 register (Register 107). In this mode, the device will power off all devices except for the primary I2C interface, waking only the accelerometer at fixed intervals to take a single measurement. The frequency of wake-ups can be configured with LP_WAKE_CTRL as shown below:

For further information regarding the MPU-60X0's power modes, please refer to Register 107.

Return Current wake frequency

See MPU6050_RA_PWR_MGMT_2

void **setWakeFrequency**(uint8_t *frequency*)
Set wake frequency in Accel-Only Low Power Mode.

See MPU6050_RA_PWR_MGMT_2

Parameters

- `frequency`: New wake frequency

bool **getStandbyXAccelEnabled**()
Get X-axis accelerometer standby enabled status.

If enabled, the X-axis will not gather or report data (or use power).

Return Current X-axis standby enabled status

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_XA_BIT

void **setStandbyXAccelEnabled**(bool *enabled*)
Set X-axis accelerometer standby enabled status.

See *getStandbyXAccelEnabled()*

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_XA_BIT

Parameters

- `New`: X-axis standby enabled status

bool **getStandbyYAccelEnabled**()
Get Y-axis accelerometer standby enabled status.

If enabled, the Y-axis will not gather or report data (or use power).

Return Current Y-axis standby enabled status

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_YA_BIT

void **setStandbyYAccelEnabled**(bool *enabled*)
Set Y-axis accelerometer standby enabled status.

See *getStandbyYAccelEnabled()*

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_YA_BIT

Parameters

- `New`: Y-axis standby enabled status

bool **getStandbyZAccelEnabled**()
Get Z-axis accelerometer standby enabled status.

If enabled, the Z-axis will not gather or report data (or use power).

Return Current Z-axis standby enabled status

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_ZA_BIT

void **setStandbyZAccelEnabled**(bool *enabled*)
Set Z-axis accelerometer standby enabled status.

See *getStandbyZAccelEnabled()*

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_ZA_BIT

Parameters

- `New`: Z-axis standby enabled status

bool **getStandbyXGyroEnabled**()
Get X-axis gyroscope standby enabled status.

If enabled, the X-axis will not gather or report data (or use power).

Return Current X-axis standby enabled status

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_XG_BIT

void **setStandbyXGyroEnabled**(bool *enabled*)
Set X-axis gyroscope standby enabled status.

See *getStandbyXGyroEnabled()*

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_XG_BIT

Parameters

- `New`: X-axis standby enabled status

bool **getStandbyYGyroEnabled**()
Get Y-axis gyroscope standby enabled status.

If enabled, the Y-axis will not gather or report data (or use power).

Return Current Y-axis standby enabled status

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_YG_BIT

void **setStandbyYGyroEnabled**(bool *enabled*)
Set Y-axis gyroscope standby enabled status.

See *getStandbyYGyroEnabled()*

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_YG_BIT

Parameters

- `New`: Y-axis standby enabled status

bool **getStandbyZGyroEnabled**()
Get Z-axis gyroscope standby enabled status.

If enabled, the Z-axis will not gather or report data (or use power).

Return Current Z-axis standby enabled status

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_ZG_BIT

void **setStandbyZGyroEnabled**(bool *enabled*)
Set Z-axis gyroscope standby enabled status.

See *getStandbyZGyroEnabled()*

See MPU6050_RA_PWR_MGMT_2

See MPU6050_PWR2_STBY_ZG_BIT

Parameters

- `New`: Z-axis standby enabled status

uint16_t **getFIFOCount**()
Get current FIFO buffer size.

This value indicates the number of bytes stored in the FIFO buffer. This number is in turn the number of bytes that can be read from the FIFO buffer and it is directly proportional to the number of samples available given the set of sensor data bound to be stored in the FIFO (register 35 and 36).

Return Current FIFO buffer size

uint8_t **getFIFOByte**()
Get byte from FIFO buffer.

This register is used to read and write data from the FIFO buffer. Data is written to the FIFO in order of register number (from lowest to highest). If all the FIFO enable flags (see below) are enabled and all External *Sensor* Data registers (Registers 73 to 96) are associated with a Slave device, the contents of registers 59 through 96 will be written in order at the Sample Rate.

The contents of the sensor data registers (Registers 59 to 96) are written into the FIFO buffer when their corresponding FIFO enable flags are set to 1 in FIFO_EN (Register 35). An additional flag for the sensor data registers associated with I2C Slave 3 can be found in I2C_MST_CTRL (Register 36).

If the FIFO buffer has overflowed, the status bit FIFO_OFLOW_INT is automatically set to 1. This bit is located in INT_STATUS (Register 58). When the FIFO buffer has overflowed, the oldest data will be lost and new data will be written to the FIFO.

If the FIFO buffer is empty, reading this register will return the last byte that was previously read from the FIFO until new data is available. The user should check FIFO_COUNT to ensure that the FIFO buffer is not read when empty.

Return Byte from FIFO buffer

void **setFIFOByte**(uint8_t *data*)
Write byte to FIFO buffer.

See *getFIFOByte()*

See MPU6050_RA_FIFO_R_W

void **getFIFOBytes**(uint8_t **data*, uint8_t *length*)

uint8_t **getDeviceID**()
Get Device ID.

This register is used to verify the identity of the device (0b110100, 0x34).

Return Device ID (6 bits only! should be 0x34)

See MPU6050_RA_WHO_AM_I

See MPU6050_WHO_AM_I_BIT

See MPU6050_WHO_AM_I_LENGTH

void **setDeviceID**(uint8_t *id*)
Set Device ID.

Write a new ID into the WHO_AM_I register (no idea why this should ever be necessary though).

See *getDeviceID()*

See MPU6050_RA_WHO_AM_I

See MPU6050_WHO_AM_I_BIT

See MPU6050_WHO_AM_I_LENGTH

Parameters

- `id`: New device ID to set.

uint8_t **getOTPBankValid**()

void **setOTPBankValid**(bool *enabled*)

int8_t **getXGyroOffsetTC**()

void **setXGyroOffsetTC**(int8_t *offset*)

int8_t **getYGyroOffsetTC**()

void **setYGyroOffsetTC**(int8_t *offset*)

int8_t **getZGyroOffsetTC**()

void **setZGyroOffsetTC**(int8_t *offset*)

int8_t **getXFineGain**()

void **setXFineGain**(int8_t *gain*)

int8_t **getYFineGain**()

void **setYFineGain**(int8_t *gain*)

int8_t **getZFineGain**()

void **setZFineGain**(int8_t *gain*)

int16_t **getXAccelOffset**()

void **setXAccelOffset**(int16_t *offset*)

int16_t **getYAccelOffset**()

void **setYAccelOffset**(int16_t *offset*)

int16_t **getZAccelOffset**()

void **setZAccelOffset**(int16_t *offset*)

int16_t **getXGyroOffset**()

void **setXGyroOffset**(int16_t *offset*)

int16_t **getYGyroOffset**()

void **setYGyroOffset**(int16_t *offset*)

int16_t **getZGyroOffset**()

void **setZGyroOffset**(int16_t *offset*)

bool **getIntPLLReadyEnabled**()

void **setIntPLLReadyEnabled**(bool *enabled*)

bool **getIntDMPEnabled**()

void **setIntDMPEnabled**(bool *enabled*)

bool **getDMPInt5Status**()

bool **getDMPInt4Status**()

bool **getDMPInt3Status**()

bool **getDMPInt2Status**()

bool **getDMPInt1Status**()

bool **getDMPInt0Status**()

bool **getIntPLLReadyStatus**()

bool **getIntDMPStatus**()

bool **getDMPEnabled**()

void **setDMPEnabled**(bool *enabled*)

void **resetDMP**()

void **setMemoryBank**(uint8_t *bank*, bool *prefetchEnabled* = false, bool *userBank* = false)

void **setMemoryStartAddress**(uint8_t *address*)

uint8_t **readMemoryByte**()

void **writeMemoryByte**(uint8_t *data*)

void **readMemoryBlock**(uint8_t **data*, uint16_t *dataSize*, uint8_t *bank* = 0, uint8_t *address* = 0)

bool **writeMemoryBlock**(**const** uint8_t **data*, uint16_t *dataSize*, uint8_t *bank* = 0, uint8_t *address* = 0, bool *verify* = true, bool *useProgMem* = false)

bool **writeProgMemoryBlock**(**const** uint8_t **data*, uint16_t *dataSize*, uint8_t *bank* = 0, uint8_t *address* = 0, bool *verify* = true)

bool **writeDMPConfigurationSet**(**const** uint8_t **data*, uint16_t *dataSize*, bool *useProgMem* = false)

bool **writeProgDMPConfigurationSet**(**const** uint8_t **data*, uint16_t *dataSize*)

uint8_t **getDMPConfig1**()

void **setDMPConfig1**(uint8_t *config*)

uint8_t **getDMPConfig2**()

void **setDMPConfig2**(uint8_t *config*)

Private Members

uint8_t **devAddr**

uint8_t **buffer**[14]

struct pb_bytes_array_s

Public Members

pb_size_t **size**

pb_byte_t **bytes**[1]

struct pb_callback_s

Public Members

bool (***decode**) (*pb_istream_t* *stream, **const** pb_field_t *field, void ***arg*)

bool (***encode**) (*pb_ostream_t* *stream, **const** pb_field_t *field, void ***const** **arg*)

union pb_callback_s::@1 pb_callback_s::funcs

void ***arg**

struct pb_extension_s

Public Members

const *pb_extension_type_t* ***type**

void ***dest**

pb_extension_t ***next**

bool **found**

struct pb_extension_type_s

Public Members

bool (***decode**) (*pb_istream_t* *stream, *pb_extension_t* *extension, uint32_t tag, *pb_wire_type_t* wire_type)

bool (***encode**) (*pb_ostream_t* *stream, **const** *pb_extension_t* *extension)

const void ***arg**

struct pb_field_iter_s

Public Members

const pb_field_t ***start**

const pb_field_t ***pos**

unsigned **required_field_index**

void ***dest_struct**

void ***pData**

void ***pSize**

struct pb_field_s

Public Members

pb_size_t **tag**

pb_type_t **type**

pb_size_t **data_offset**

pb_ssize_t **size_offset**

pb_size_t **data_size**

pb_size_t **array_size**

const void ***ptr**

struct pb_istream_s

Public Members

bool (***callback**) (*pb_istream_t* *stream, *pb_byte_t* *buf, size_t count)

void ***state**

size_t **bytes_left**

const char ***errmsg**

struct pb_ostream_s

Public Members

bool (***callback**) (*pb_ostream_t* *stream, **const** *pb_byte_t* *buf, size_t count)

void ***state**

size_t **max_size**

size_t **bytes_written**

const char ***errmsg**

struct PoseMinimizer

Inherits from *Functor< double >*

Public Functions

PoseMinimizer (int *numberOfSensors* = 4)

Default amount of sensors needed for Eigen templated structure.

Parameters

- `numberOfSensors`: you can however choose any number of sensors here

int **operator()** (**const** VectorXd &*x*, VectorXd &*fvec*) **const**
This is the function that is called in each iteration.

Return

Parameters

- `x`: the pose vector (3 rotational 3 translational parameters)
- `fvec`: the error function (the difference between the sensor positions)

void **getRTmatrix** (VectorXd &*x*, Matrix4d &*RT*)
Constructs a Transform matrix from the given pose vector.

Parameters

- `x`: the pose vector
- `RT`: the corresponding transform

void **getRTmatrix** (Matrix4d &*RT*)
Constructs a Transform matrix from the optimized pose.

Parameters

- `RT`: the corresponding transform

void **getTFtransform** (VectorXd &*x*, tf::Transform &*tf*)
Constructs a tf transform from the given pose vector.

Parameters

- `x`: the pose vector
- `tf`: the corresponding tf transform

Public Members

VectorXd **pose**

MatrixXd **pos3D_A**

MatrixXd **pos3D_B**

int **numberOfSensors** = 4

class PROTO_LOVE

Public Functions

PROTO_LOVE()

void **clearProtos**()

bool **decode_config_Proto**(*pb_byte_t* *buffer, size_t rcvd_msg_len)

bool **decode_command_Proto**(*pb_byte_t* *buffer, size_t rcvd_msg_len)

bool **encode_trackedObjConfig**(uint32_t *ip*, uint16_t *cmndPort_l*, *pb_byte_t* *buffer, size_t &msg_len)

bool **encode_loggingObject**(**const** char *msg, *pb_byte_t* *buffer, size_t &msg_len)

bool **encode_imuObjConfig**(*Quaternion* &q, *VectorInt16* &acc, *VectorFloat* &gravity, *pb_byte_t* *buffer, size_t &msg_len)

Public Members

bool **enable_logging** = true

DarkRoomProtobuf_loggingObject **loggingObjMsg**

DarkRoomProtobuf_commandObject **commandObjMsg**

DarkRoomProtobuf_configObject **configObjMsg**

DarkRoomProtobuf_imuObject **imuObjMsg**

class Quaternion

Public Functions

Quaternion()

Quaternion(float *nw*, float *nx*, float *ny*, float *nz*)

Quaternion **getProduct**(*Quaternion* *q*)

Quaternion **getConjugate**()

float **getMagnitude**()

void **normalize**()

Quaternion **getNormalized**()

Public Members

float **w**

float **x**

float **y**

float **z**

class Sensor

Public Types

enum ANGLE_TYPE
Values:

HORIZONTAL = 0

VERTICAL = 1

Public Functions

Sensor()

void **update**(bool *lighthouse*, int *type*, unsigned short *timestamp*, double *angle*)
Updates the angles for the respective lighthouse.

Parameters

- `lighthouse`: for which lighthouse
- `type`: #ANGLE_TYPE
- `timestamp`: millisecond timestamp
- `angle`: ligthhouse angle

void **switchLighthouses**(bool *flag*)
Switches lighthouses, such that angles for lighthouse 1 belong to lighthouse 2.

Parameters

- `flag`: true/false

bool **getSwitchLighthouses**()
Returns if the lighthouses are switched.

Return true/false

void **get**(bool *lighthouse*, Vector2d &*angles*)
Get the recent angles.

Parameters

- `lighthouse`: of this lighthouse
- `angles`: Vector(VERTICAL, HORIZONTAL)

void **get**(bool *lighthouse*, Vector2d &*angles*, *Vector2s* &*timestamps*)
Get the recent angles and timestamps.

Parameters

- `lighthouse`: of this lighthouse
- `angles`: Vector(VERTICAL, HORIZONTAL)
- `timestamps`: in milliseconds

void **get**(bool *lighthouse*, double &*elevation*, double &*azimuth*)
Get the recent angles.

Parameters

- `lighthouse`: of this lighthouse
- `elevation`: angle
- `azimuth`: angle

void **get**(bool *lighthouse*, vector<double> &*elevations*, vector<double> &*azimuths*)
pushes the recent angles into the vectors

Parameters

- `lighthouse`: of this lighthouse
- `elevations`: vector<angle>
- `azimuths`: vector<angle>

void **set**(bool *lighthouse*, Vector3d &*position3D*)
Set the relative 3d position, wrt to a lighthouse.

Parameters

- `lighthouse`:
- `position3D`:

bool **getPosition3D**(Vector3d &*position3D*)
Gets the 3d position of a sensor.

Return

Parameters

- `position3D`:

bool **get**(bool *lighthouse*, Vector3d &*position3D*)
Gets the relative 3d position of a sensor wrt to a lighthouse.

Return

Parameters

- `lighthouse`:
- `position3D`:

void **set**(Vector3d &*position3D*)
Set the absolute 3d position, wrt to a world frame.

Parameters

- `lighthouse`:
- `position3D`:

double **getDistance**(bool *lighthouse*)
Returns the euclidean distance to a lighthouse in meter.

Return

Parameters

- `lighthouse`:

bool **isActive**(bool *lighthouse*)
A sensor is considered active, when it has been updated less than a certain amount of time ago.

Return active/inactive

Parameters

- `lighthouse`:

void **setRelativeLocation**(Vector3d *relative_location*)
Sets the sensors relative location on the tracked object.

Parameters

- `relative_location`: the relative 3D position

void **getRelativeLocation**(Vector3d &*relative_location*)
Gets the sensors relative location on the tracked object.

Parameters

- `relative_location`: the relative 3D position

void **getRelativeLocation**(vector<Vector3d> &*relative_locations*)
pushes the sensors relative location on the tracked object

Parameters

- `relative_locations`: into this vector

Private Members

bool **m_switch** = true

Vector3d **m_relative_location**

Vector3d **m_position3D**

Vector3d **m_relativePosition3D**[NUMBER_OF_LIGHTHOUSES]

Vector3d **m_relativeOrigin3D**[NUMBER_OF_LIGHTHOUSES]

pair<unsigned short, double> **m_angles_horizontal**[NUMBER_OF_LIGHTHOUSES]

pair<unsigned short, double> **m_angles_vertical**[NUMBER_OF_LIGHTHOUSES]

ros::Time **m_angleUpdateTime**[NUMBER_OF_LIGHTHOUSES]

mutex **m_lockMutex**

class SPISlaveClass

Public Functions

SPISlaveClass()

~SPISlaveClass()

void **begin**()

void **setData**(uint8_t **data*, size_t *len*)

void **setData**(**const** char **data*)

void **setStatus**(uint32_t *status*)

void **onData**(*SpiSlaveDataHandler* *cb*)

void **onDataSent**(*SpiSlaveSentHandler* *cb*)

void **onStatus**(*SpiSlaveStatusHandler* *cb*)

void **onStatusSent**(*SpiSlaveSentHandler* *cb*)

Protected Functions

void **_data_rx**(uint8_t **data*, uint8_t *len*)

void **_status_rx**(uint32_t *data*)

void **_data_tx**(void)

void **_status_tx**(void)

Protected Attributes

SpiSlaveDataHandler **_data_cb**

SpiSlaveStatusHandler **_status_cb**

SpiSlaveSentHandler **_data_sent_cb**

SpiSlaveSentHandler **_status_sent_cb**

Protected Static Functions

void **_s_data_rx**(void **arg*, uint8_t **data*, uint8_t *len*)

void **_s_status_rx**(void **arg*, uint32_t *data*)

void **_s_data_tx**(void **arg*)

void **_s_status_tx**(void **arg*)

class StarRating

Public Types

enum **EditMode**

Values:

Editable

ReadOnly

Public Functions

StarRating(int *starCount* = 1, int *maxStarCount* = 5)

void **paint**(QPainter **painter*, **const** QRect &*rect*, **const** QPalette &*palette*, *EditMode mode*) **const**

QSize **sizeHint**() **const**

int **starCount**() **const**

int **maxStarCount**() **const**

void **setStarCount**(int *starCount*)

void **setMaxStarCount**(int *maxStarCount*)

Private Members

QPolygonF **starPolygon**

QPolygonF **diamondPolygon**

int **myStarCount**

int **myMaxStarCount**

struct **Sweep**

Public Members

uint32_t **sweepDuration**

uint16_t **lighthouse**

uint16_t **id**

bool **vertical**

class **TrackedObject**

Public Functions

TrackedObject()

~TrackedObject()

bool **connectWifi**(**const** char *_IP_, int _sensor_port_, int _command_port_, int _logging_port_, bool _configure_ = true)

This function attempts to establish the connection to the tracked object via wifi.

Return success (if configure is false, this returns true even if there is no data coming from clientbool lighthouse)

Parameters

- `IP`: IP of the object
- `sensor_port`: port to be used for sensor value
- `command_port`: port to be used for commands
- `logging_port`: port to be used for logging
- `configure`: if true sends our ports to client and waits for a logging message

void **startReceiveData**(bool _start_)

void **startTracking**(bool _start_)

toggles tracking (starts reception of sensor data and triangulation threads)

Parameters

- `start`: toggle flag

void **startCalibration**(bool _start_)

Toggles calibration thread.

Parameters

- `start`: bool

void **showRays**(bool _show_)

Toggles visualization of ligthhouse rays.

Parameters

- `show`: bool

void **map_lighthouse_id**(bool _switchID_)

This function switches the lighthouse ids for all objects.

bool **distanceEstimation**(bool _lighthouse_, vector<int> *_specificIds_ = nullptr)

Estimates the sensor distances of all active sensors (or a vector of specified sensor ids) using the known (ie.

calibrated) relative distance between the sensors and the lighthouse angles

Return

Parameters

- `lighthouse`: for which lighthouse
- `specificIds`: if defined, waits until the specified sensors become active

bool **poseEstimation**(tf::Transform &*tf*)
Estimates the pose correction between ligthhouse 1 and 2, such that the squared distances between sensor positions estimated for both lighthouses is minimized.

Return success

Parameters

- `tf`: the corrective transform for lighthouse 2

bool **record**(bool *start*)
Triggers recording of sensor data to a sensor.log file.

Return success (if writing to the file is possible)

Parameters

- `start`:

bool **readConfig**(string *filepath*)
Reads a yaml tracked object file.

Return success

Parameters

- `filepath`: to

bool **writeConfig**(string *filepath*)
Writes a tracked object to file.

Return success

Parameters

- `filepath`:

bool **rebootESP**()
Sends a command which triggers ESP reboot.

Return success

bool **toggleMPU6050**(bool *toggle*)
Toggles the IMU *MPU6050*.

Return successully sent

Parameters

- `toggle`: true/false

bool **toggleTracking**(bool *toggle*)
Toggles the tracking on fpga.

Return successully sent

Parameters

- `toggle`: true/false

Public Static Attributes

int **trackeObjectInstance** = 0

Private Types

enum MESSAGE_ID

Values:

TRIANGULATED = 0

DISTANCE = 1

RAY = 2

Private Functions

void **receiveSensorData**()

a unique object instance (helps with unique rviz marker ids)

Continuesly receiving, decoding and updating the sensor data

void **receiveImuData**()

Continuesly receiving imu data.

void **trackSensors**()

Triangulates the sensor positions (the transform between lighthouse 1 and 2 needs to be known, otherwise the triangulated position is not correct)

void **calibrate**()

Measures triangulated sensor locations for 30 seconds.

Calculates mean sensor locations and generates relative sensor positions which are saved to a yaml file

void **publishRelativeFrame**()

Publishes information about the relative frame of the Object (as estimated via calibration)

bool **triangulateFromLighthousePlanes**(Vector2d *&angles0*, Vector2d *&angles1*, Vector3d *&triangulated_position*, Vector3d *&ray0*, Vector3d *&ray1*)

This function triangulates the position of a sensor using the horizontal and vertical angles from two ligthouses via planes.

Return success (this can fail, if we don't get the pose of the lighthouses via tf)

Parameters

- `angles0`: vertical/horizontal angles form first lighthouse
- `angles1`: vertical/horizontal angles form second lighthouse
- `triangulated_position`: the triangulated position
- `ray0`: ligthhouse ray
- `ray1`: ligthhouse ray

bool **getTransform**(**const** char **from*, **const** char **to*, Matrix4d *&transform*)
Queries the tf listener for the specified transform.

Return true if available

Parameters

- `from`: we whant the transformation from this frame
- `to`: another frame
- `transform`: the transform if available

void **clearAll**()
clear all markers

int **getMessageID**(int *type*, int *sensor*, bool *lighthouse* = false)
Returns a unique id for #MESSAGE_ID sensor and lighthouse.

Return a unique id

Parameters

- `type`: the message type #MESSAGE_ID
- `sensor`: the sensor id
- `lighthouse`: the lighthouse

void **publishMesh**(Vector3d *&pos*, Vector4d *&orientation*, **const** char **modelname*, **const** char **frame*, **const** char **ns*, int *message_id*, int *duration*)
Publishes a mesh visualization marker.

Parameters

- `pos`: at this position
- `orientation`: with this orientation
- `modelname`: name of the mesh.dae
- `frame`: in this frame
- `ns`: namespace
- `message_id`: unique id
- `duration`: in seconds

void **publishSphere**(Vector3d *&pos*, **const** char **frame*, **const** char **ns*, int *message_id*, *COLOR color*, float *radius* = 0.01, int *duration* = 0)
Publishes a sphere visualization marker.

Parameters

- `pos`: at this positon
- `frame`: in this frame
- `ns`: namespace
- `message_id`: a unique id
- `rgda`: rgb color (0-1) plus transparancy

- `duration`: for this duration in seconds (0=forever)

void **publishCube**(Vector3d &*pos*, Vector4d &*quat*, **const** char **frame*, **const** char **ns*, int *message_id*, *COLOR color*, float *radius* = 0.01, int *duration* = 0)

Publishes a cube visualization marker.

Parameters

- `pos`: at this positon
- `quat`: with this orientation
- `frame`: in this frame
- `ns`: namespace
- `message_id`: a unique id
- `rgda`: rgb color (0-1) plus transparancy
- `duration`: for this duration in seconds (0=forever)

void **publishRay**(Vector3d &*pos*, Vector3d &*dir*, **const** char **frame*, **const** char **ns*, int *message_id*, *COLOR color*, int *duration* = 0)

Publishes a ray visualization marker.

Parameters

- `pos`: at this positon
- `dir`: direction
- `frame`: in this frame
- `message_id`: a unique id
- `ns`: namespace
- `rgda`: rgb color (0-1) plus transparancy
- `duration`: for this duration in seconds (0=forever)

Private Members

ros::NodeHandlePtr **nh**

boost::shared_ptr<ros::AsyncSpinner> **spinner**

ros::Publisher **visualization_pub**

tf::Transform **relativeFrame**

tf::TransformListener **tf_listener**

tf::TransformBroadcaster **tf_broadcaster**

int **objectID** = 0

string **path**

string **name** = "bastiisdoff"

string **mesh** = "pimmel"

vector<*Eigen*::Vector3f> **object**

double **axis_offset** = -0.015

Vector3d **poseIMU**

int **sensordata_port**

UDPSocketPtr **sensor_socket**

UDPSocketPtr **command_socket**

UDPSocketPtr **logging_socket**

UDPSocketPtr **imu_socket**

boost::shared_ptr<thread> **sensor_thread** = nullptr

boost::shared_ptr<thread> **tracking_thread** = nullptr

boost::shared_ptr<thread> **calibrate_thread** = nullptr

boost::shared_ptr<thread> **imu_thread** = nullptr

bool **receiveData** = false

bool **tracking** = false

bool **calibrating** = false

bool **connected** = false

bool **rays** = false

bool **recording** = false

bool **publishingRelativeFrame** = false

map<int, *Sensor*> **sensors**

Vector3d **origin**

Vector4d **pose**

Vector4d **zero_pose**

bool **m_switch** = false

ofstream **file**

class TrackedObjectDelegate

Inherits from QStyledItemDelegate

Public Functions

TrackedObjectDelegate(QWidget *parent = 0)

void **paint**(QPainter *painter, **const** QStyleOptionViewItem &option, **const** QModelIndex &index) **const**

QSize **sizeHint**(**const** QStyleOptionViewItem &option, **const** QModelIndex &index) **const**

QWidget ***createEditor**(QWidget *parent, **const** QStyleOptionViewItem &option, **const** QModelIndex &index) **const**

void **setEditorData**(QWidget *editor, **const** QModelIndex &index) **const**

void **setModelData**(QWidget *editor, QAbstractItemModel *model, **const** QModelIndex &index) **const**

Private Slots

void **commitAndCloseEditor**()

class TrackedObjectEditor
Inherits from QWidget

Public Functions

TrackedObjectEditor(QWidget *parent = 0)

QSize **sizeHint**() **const**

void **setStarRating**(**const** *StarRating* &starRating)

StarRating **starRating**()

Signal

signal TrackedObjectEditor::editingFinished

Protected Functions

void **paintEvent**(QPaintEvent *event)

void **mouseMoveEvent**(QMouseEvent *event)

void **mouseReleaseEvent**(QMouseEvent *event)

Private Functions

int **starAtPosition**(int x)

Private Members

StarRating **myStarRating**

class UDPSocket

Public Functions

UDPSocket(**const** char *server_IP, int server_port, **const** char *client_IP, int client_port, bool exclusive = true)

Creates a socket on the given server_IP and server_port and sets up the "connection" with the client.

Because it is UDP, there is no handshake, the socket just sends and listens to packages from the client_IP and client_port

Parameters

- `server_IP`: the server socket IP

- `server_port`: the server socket port
- `client_IP`: the client to send and receive UDP packets to/from
- `client_port`: the client port
- `exclusive`: receive exclusively packages from client

UDPSocket (**const** char **client_IP*, int *client_port*, bool *exclusive* = true)
Tries to guess the users IP and sets up the socket on Port 8000 and "connects" to client_IP on client_port.

Parameters

- `client_IP`: the client to send and receive UDP packets to/from
- `client_port`: the client port
- `exclusive`: receive exclusively packages from client

UDPSocket (**const** char **client_IP*, int *client_port*, int *server_port*, bool *exclusive* = true)

UDPSocket (int *port*, bool *broadcaster*)
Creates a broadcast socket on port.

Parameters

- `port`:

~UDPSocket ()

bool **receiveSensorData** (unsigned short *&timestamp*, int *&id*, int *&lighthouse*, int *&rotor*, uint *&sweepDuration*)
Receive 32-bit sensor UDP and disect into values.

Return success

Parameters

- `timestamp`:
- `id`: sensor id
- `lighthouse`: for which lighthouse
- `rotor`: which rotor (horizontal/vertical)
- `sweepDuration`: the sweep duration

bool **receiveSensorData** (unsigned short *&timestamp*, vector<uint> *&id*, vector<uint> *&lighthouse*, vector<uint> *&rotor*, vector<uint> *&sweepDuration*)
Receive multiple 32-bit sensor values in a single UDP and disect into values.

Return success

Parameters

- `timestamp`:
- `id`: sensor id
- `lighthouse`: for which lighthouse
- `rotor`: which rotor (horizontal/vertical)

- `sweepDuration`: the sweep duration

template <typename T**>**
bool **`receiveMessage`**(T *&message*)
Receives a google protobuf message from the client.

Return success

Parameters

- `message`: reference to protbuf message

template <typename T**>**
bool **`sendMessage`**(T *&message*)
Sends a google protobuf message to the client.

Return success

Parameters

- `message`: reference to protbuf message

template <typename T**>**
bool **`broadcastMessage`**(T *&message*)
broadcasts a google protobuf message

Return success

Parameters

- `message`: reference to protbuf message

Public Members

pair<uint32_t, string> **`myIP`**

Private Functions

bool **`setTimeOut`**(int *usecs*)
Sets the UDP package receive and send timeout.

Return success

Parameters

- `usecs`: time in microseconds

bool **`whatsMyIP`**(string *&IP*)
Tries to guess your IP.

Parameters

- `ip`: your IP
- `success`: (fails if I can't find a valid IP for wifi or ethernet adapter)

bool **receiveUDP**()
receive from anyone

Return success

bool **receiveUDPFromClient**()
receive from client

Return success

bool **sendUDPToClient**()
send to client

Return success

bool **broadcastUDP**()
broadcast

Return success

Private Members

bool **initialized** = false

int **sockfd**

struct sockaddr_in **server_addr**

struct sockaddr_in **broadcast_addr**

struct sockaddr_in **client_addr**

socklen_t **client_addr_len**

socklen_t **server_addr_len**

socklen_t **broadcast_addr_len**

ssize_t **numbytes**

struct addrinfo ***servinfo**

char **buf**[MAXBUFLENGTH]

bool **exclusive**

class VectorFloat

Public Functions

VectorFloat()

VectorFloat(float *nx*, float *ny*, float *nz*)

float **getMagnitude**()

void **normalize**()

VectorFloat **getNormalized**()

void **rotate**(*Quaternion* *q)

VectorFloat **getRotated**(*Quaternion* *q)

Public Members

float **x**

float **y**

float **z**

class VectorInt16

Public Functions

VectorInt16()

VectorInt16(int16_t *nx*, int16_t *ny*, int16_t *nz*)

float **getMagnitude**()

void **normalize**()

VectorInt16 **getNormalized**()

void **rotate**(*Quaternion* *q)

VectorInt16 **getRotated**(*Quaternion* *q)

Public Members

int16_t **x**

int16_t **y**

int16_t **z**

class WIFI_LOVE

Public Functions

WIFI_LOVE(**const** char **ssid*, **const** char **passwd*, IPAddress &*broadcastIP*)

uint16_t **getCmndPort**()

uint32_t **getLocalIP**()

int **printWifiStatus**()

int **initWifi**()

int **initUDPSockets**()

int **fmsgTest_s**()

int **fmsgBroadcast_s**(**const** uint8_t *_buffer_, size_t _size_)

int **fmsgLogging_s**(**const** uint8_t *_buffer_, size_t _size_)

int **fmsgSensorDataT_s**(**const** uint8_t *_buffer_, size_t _size_)

int **fmsgSensorData_s**(**const** uint8_t *_buffer_, size_t _size_)

int **fmsgImuData_s**(**const** uint8_t *_buffer_, size_t _size_)

bool **receiveCommand**()

bool **receiveConfig**()

bool **sendImuData**(*Quaternion* &_q_, *VectorInt16* &_acc_, *VectorFloat* &_gravity_)

int **getConnectionStatus**()

void **checkHostConfig**()

void **printIP**(uint32_t _ip_)

Public Members

int **LoveStatus** = WL_IDLE_STATUS

char **ssid**[100]

char **pass**[100]

bool **timeout** = false

uint16_t **sensorPort** = 0

uint16_t **commandPort** = 4210

uint16_t **logginPort** = 0

uint16_t **configPort** = 8001

uint16_t **imuPort** = 0

IPAddress **hostIP**

IPAddress **broadcastIP**

WiFiUDP **UDP**

PROTO_LOVE **protoLove**

uint32_t **command**

size_t **msg_len**

pb_byte_t **buffer**[512] = {0}

namespace Eigen

namespace std

file **fifo.h**

Defines

FIFO_init(fifo) { fifo.mRead = 0; fifo.mWrite = 0;}

FIFO_available(fifo) (fifo.mRead != fifo.mWrite)

FIFO_read(fifo, size) (\ (FIFO_available(fifo)) ? \ fifo.mBuffer[(fifo.mRead = ((fifo.mRead + 1) & (size -1)))] : 0 \ )

FIFO_write(fifo, data, size) { \ uint8_t temp = ((fifo.mWrite +1)& (size -1)); \ if(temp != fifo.mRead) { \ fifo.mBuffer[temp] = data; \ fifo.mWrite = temp; \ } \ }

FIFO32_read(fifo) FIFO_read(fifo, 32)

FIFO32_write(fifo, data) FIFO_write(fifo, data, 32)

Typedefs

typedef struct *Sweep* **Sweep**

typedef struct *_FIFO128sweep* **FIFO32sweep**

typedef struct *_FIFO128t* **FIFO32t**

file **helper_3dmath.h**

file **hspi_slave.c**
#include "hspi_slave.h"#include "esp8266_peri.h"#include "ets_sys.h"

Functions

void ICACHE_RAM_ATTR _hspi_slave_isr_handler(void * arg)

void **hspi_slave_begin**(uint8_t *status_len*, void **arg*)

void **hspi_slave_setStatus**(uint32_t *status*)

void **hspi_slave_setData**(uint8_t **data*, uint8_t *len*)

void **hspi_slave_onData**(void (**rxd_cb*)) void *, uint8_t *, uint8_t

void **hspi_slave_onDataSent**(void (**txd_cb*)) void *

void **hspi_slave_onStatus**(void (**rxs_cb*)) void *, uint32_t

void **hspi_slave_onStatusSent**(void (**txs_cb*)) void *

Variables

void (***_hspi_slave_rx_data_cb**) (void *arg, uint8_t *data, uint8_t len) = NULL

void (***_hspi_slave_tx_data_cb**) (void *arg) = NULL

void (***_hspi_slave_rx_status_cb**) (void *arg, uint32_t data) = NULL

void (***_hspi_slave_tx_status_cb**) (void *arg) = NULL

uint8_t **_hspi_slave_buffer**[33]

file **hspi_slave.h**
#include "Arduino.h"

Functions

void **hspi_slave_begin** (uint8_t *status_len*, void **arg*)

void **hspi_slave_setStatus** (uint32_t *status*)

void **hspi_slave_setData** (uint8_t **data*, uint8_t *len*)

void **hspi_slave_onData** (void (**rxd_cb*)) void *, uint8_t *, uint8_t

void **hspi_slave_onDataSent** (void (**txd_cb*)) void *

void **hspi_slave_onStatus** (void (**rxs_cb*)) void *, uint32_t

void **hspi_slave_onStatusSent** (void (**txs_cb*)) void *

file **I2Cdev.cpp**
#include "I2Cdev.h"

file **I2Cdev.h**

Defines

I2CDEV_IMPLEMENTATION I2CDEV_ARDUINO_WIRE

I2CDEV_IMPLEMENTATION_WARNINGS

I2CDEV_ARDUINO_WIRE 1

I2CDEV_BUILTIN_NBWIRE 2

I2CDEV_BUILTIN_FASTWIRE 3

I2CDEV_I2CMASTER_LIBRARY 4

I2CDEV_DEFAULT_READ_TIMEOUT 1000

file **lighthouse.pb.c**
#include "lighthouse.pb.h"

Variables

const pb_field_t **DarkRoomProtobuf_configObject_fields**[5] = { PB_FIELD(1, UINT32 , REQUIRED, STATIC , F

const pb_field_t **DarkRoomProtobuf_trackedObjectConfig_fields**[3] = { PB_FIELD(1, UINT32 , REQUIRED, S

const pb_field_t **DarkRoomProtobuf_loggingObject_fields**[2] = { PB_FIELD(1, STRING , OPTIONAL, STATIC ,

const pb_field_t **DarkRoomProtobuf_commandObject_fields**[2] = { PB_FIELD(1, INT32 , REQUIRED, STATIC , F

const pb_field_t **DarkRoomProtobuf_imuObject_fields**[5] = { PB_FIELD(1, INT32 , REQUIRED, STATIC , FIRST,

file **lighthouse.pb.h**
#include "pb.h"

Defines

DarkRoomProtobuf_configObject_init_default {0, 0, 0, 0}

DarkRoomProtobuf_trackedObjectConfig_init_default {0, 0}

DarkRoomProtobuf_loggingObject_init_default {false, ""}

DarkRoomProtobuf_commandObject_init_default {0}

DarkRoomProtobuf_imuObject_init_default {0, 0, {0, 0, 0, 0}, 0, {0, 0, 0}, 0, {0, 0, 0}}

DarkRoomProtobuf_configObject_init_zero {0, 0, 0, 0}

DarkRoomProtobuf_trackedObjectConfig_init_zero {0, 0}

DarkRoomProtobuf_loggingObject_init_zero {false, ""}

DarkRoomProtobuf_commandObject_init_zero {0}

DarkRoomProtobuf_imuObject_init_zero {0, 0, {0, 0, 0, 0}, 0, {0, 0, 0}, 0, {0, 0, 0}}

DarkRoomProtobuf_commandObject_command_tag 1

DarkRoomProtobuf_configObject_ip_tag 1

DarkRoomProtobuf_configObject_logging_port_tag 2

DarkRoomProtobuf_configObject_sensor_port_tag 3

DarkRoomProtobuf_configObject_imu_port_tag 4

DarkRoomProtobuf_imuObject_id_tag 1

DarkRoomProtobuf_imuObject_quaternion_tag 2

DarkRoomProtobuf_imuObject_acc_tag 3

DarkRoomProtobuf_imuObject_gravity_tag 4

DarkRoomProtobuf_loggingObject_message_tag 1

DarkRoomProtobuf_trackedObjectConfig_ip_tag 1

DarkRoomProtobuf_trackedObjectConfig_command_port_tag 2

DarkRoomProtobuf_configObject_size 39

DarkRoomProtobuf_trackedObjectConfig_size 17

DarkRoomProtobuf_loggingObject_size 42

DarkRoomProtobuf_commandObject_size 11

DarkRoomProtobuf_imuObject_size 79

Typedefs

typedef struct *_DarkRoomProtobuf_commandObject* **DarkRoomProtobuf_commandObject**

typedef struct *_DarkRoomProtobuf_configObject* **DarkRoomProtobuf_configObject**

typedef struct *_DarkRoomProtobuf_imuObject* **DarkRoomProtobuf_imuObject**

typedef struct *_DarkRoomProtobuf_loggingObject* **DarkRoomProtobuf_loggingObject**

typedef struct *_DarkRoomProtobuf_trackedObjectConfig* **DarkRoomProtobuf_trackedObjectConfig**

Variables

const pb_field_t **DarkRoomProtobuf_configObject_fields**[5]

const pb_field_t **DarkRoomProtobuf_trackedObjectConfig_fields**[3]

const pb_field_t **DarkRoomProtobuf_loggingObject_fields**[2]

const pb_field_t **DarkRoomProtobuf_commandObject_fields**[2]

const pb_field_t **DarkRoomProtobuf_imuObject_fields**[5]

file **logging.cpp**
#include "logging.h"

Functions

static void **printWiFiUDP**()

static void **printSerial**()

static void **setLoggingLevel**(*TLogLevel newLevel*)

static void **setFlushInterface**(*FlushInterface in*)

static *LogString* ***createLogString**(char **input*, *TLogLevel level*)

static void **deleteLogString**()

static void **appendLogString**(**const** char ***const** *append*)

static *TLogLevel* **GetReportingLevel**(void)

static void **GetLogString**(*TLogLevel level*, **const** char **msg*)

static void **MakeldLogString**(*TLogLevel level*, **const** char **msg*, long int *number*)

static void **MakedLogString**(*TLogLevel level*, **const** char **msg*, int *number*)

static void **MakefLogString**(*TLogLevel level*, **const** char **msg*, float *number*)

Variables

}]

LogString ***logString**

TLogLevel **currentLoglevel** = logINFO

LogI **const logi** = {GetLogString, MakefLogString, MakeldLogString, MakedLogString, GetReportingLevel, setFlushInterface

file **logging.h**
#include "Arduino.h"

Defines

FOREACH_LEVEL(LEVEL) LEVEL(logERROR) \ LEVEL(logWARNING) \ LEVEL(logDEBUG) \ LEVEL(logINFO) \ LEVEL(logVERBOSE) \ LEVEL(logVERBOSE_1) \ LEVEL(logVERBOSE_2) \ LEVEL(logVERBOSE_3) \

GENERATE_ENUM(ENUM) ENUM,

GENERATE_STRING(STRING) #STRING,

LOG(level, message) if (level > (logi.GetReportingLevel()) || (false == enableLogging)); \ else logi.GetLogString(level, message)

LOG_ld(level, message, number) if (level > (logi.GetReportingLevel()) || (false == enableLogging)); \ else logi.MakeldLogString(level, message, number)

LOG_d(level, message, number) if (level > (logi.GetReportingLevel()) || (false == enableLogging)); \
else logi.MakedLogString(level, (message), (number))

LOG_f(level, message, number) if (level > (logi.GetReportingLevel()) || (false == enableLogging)); \
else logi.MakefLogString(level, message, number)

Typedefs

typedef enum *TLogLevel* **TLogLevel**

typedef enum *FlushInterface* **FlushInterface**

typedef struct *LogI* **LogI**

typedef void (***FlushI**)(void)

typedef struct *LogString* **LogString**

Enums

enum TLogLevel
Values:

enum FlushInterface
Values:

F_SERIAL = 1

F_WIFI = 2

Variables

bool **enableLogging** = true

LogI **const logi**

file **MPU6050.cpp**
#include "MPU6050.h"

file **MPU6050.h**
#include "I2Cdev.h"

Defines

MPU6050_ADDRESS_AD0_LOW 0x68

MPU6050_ADDRESS_AD0_HIGH 0x69

MPU6050_DEFAULT_ADDRESS MPU6050_ADDRESS_AD0_LOW

MPU6050_RA_XG_OFFS_TC 0x00

MPU6050_RA_YG_OFFS_TC 0x01

MPU6050_RA_ZG_OFFS_TC 0x02

MPU6050_RA_X_FINE_GAIN 0x03

MPU6050_RA_Y_FINE_GAIN 0x04

MPU6050_RA_Z_FINE_GAIN 0x05

MPU6050_RA_XA_OFFS_H 0x06

MPU6050_RA_XA_OFFS_L_TC 0x07

MPU6050_RA_YA_OFFS_H 0x08

MPU6050_RA_YA_OFFS_L_TC 0x09

MPU6050_RA_ZA_OFFS_H 0x0A

MPU6050_RA_ZA_OFFS_L_TC 0x0B

MPU6050_RA_SELF_TEST_X 0x0D

MPU6050_RA_SELF_TEST_Y 0x0E

MPU6050_RA_SELF_TEST_Z 0x0F

MPU6050_RA_SELF_TEST_A 0x10

MPU6050_RA_XG_OFFS_USRH 0x13

MPU6050_RA_XG_OFFS_USRL 0x14

MPU6050_RA_YG_OFFS_USRH 0x15

MPU6050_RA_YG_OFFS_USRL 0x16

MPU6050_RA_ZG_OFFS_USRH 0x17

MPU6050_RA_ZG_OFFS_USRL 0x18

MPU6050_RA_SMPLRT_DIV 0x19

MPU6050_RA_CONFIG 0x1A

MPU6050_RA_GYRO_CONFIG 0x1B

MPU6050_RA_ACCEL_CONFIG 0x1C

MPU6050_RA_FF_THR 0x1D

MPU6050_RA_FF_DUR 0x1E

MPU6050_RA_MOT_THR 0x1F

MPU6050_RA_MOT_DUR 0x20

MPU6050_RA_ZRMOT_THR 0x21

MPU6050_RA_ZRMOT_DUR 0x22

MPU6050_RA_FIFO_EN 0x23

MPU6050_RA_I2C_MST_CTRL 0x24

MPU6050_RA_I2C_SLV0_ADDR 0x25

MPU6050_RA_I2C_SLV0_REG 0x26

MPU6050_RA_I2C_SLV0_CTRL 0x27

MPU6050_RA_I2C_SLV1_ADDR 0x28

MPU6050_RA_I2C_SLV1_REG 0x29

MPU6050_RA_I2C_SLV1_CTRL 0x2A

MPU6050_RA_I2C_SLV2_ADDR 0x2B

MPU6050_RA_I2C_SLV2_REG 0x2C

MPU6050_RA_I2C_SLV2_CTRL 0x2D

MPU6050_RA_I2C_SLV3_ADDR 0x2E

MPU6050_RA_I2C_SLV3_REG 0x2F

MPU6050_RA_I2C_SLV3_CTRL 0x30

MPU6050_RA_I2C_SLV4_ADDR 0x31

MPU6050_RA_I2C_SLV4_REG 0x32

MPU6050_RA_I2C_SLV4_DO 0x33

MPU6050_RA_I2C_SLV4_CTRL 0x34

MPU6050_RA_I2C_SLV4_DI 0x35

MPU6050_RA_I2C_MST_STATUS 0x36

MPU6050_RA_INT_PIN_CFG 0x37

MPU6050_RA_INT_ENABLE 0x38

MPU6050_RA_DMP_INT_STATUS 0x39

MPU6050_RA_INT_STATUS 0x3A

MPU6050_RA_ACCEL_XOUT_H 0x3B

MPU6050_RA_ACCEL_XOUT_L 0x3C

MPU6050_RA_ACCEL_YOUT_H 0x3D

MPU6050_RA_ACCEL_YOUT_L 0x3E

MPU6050_RA_ACCEL_ZOUT_H 0x3F

MPU6050_RA_ACCEL_ZOUT_L 0x40

MPU6050_RA_TEMP_OUT_H 0x41

MPU6050_RA_TEMP_OUT_L 0x42

MPU6050_RA_GYRO_XOUT_H 0x43

MPU6050_RA_GYRO_XOUT_L 0x44

MPU6050_RA_GYRO_YOUT_H 0x45

MPU6050_RA_GYRO_YOUT_L 0x46

MPU6050_RA_GYRO_ZOUT_H 0x47

MPU6050_RA_GYRO_ZOUT_L 0x48

MPU6050_RA_EXT_SENS_DATA_00 0x49

MPU6050_RA_EXT_SENS_DATA_01 0x4A

MPU6050_RA_EXT_SENS_DATA_02 0x4B

MPU6050_RA_EXT_SENS_DATA_03 0x4C

MPU6050_RA_EXT_SENS_DATA_04 0x4D

MPU6050_RA_EXT_SENS_DATA_05 0x4E

MPU6050_RA_EXT_SENS_DATA_06 0x4F

MPU6050_RA_EXT_SENS_DATA_07 0x50

MPU6050_RA_EXT_SENS_DATA_08 0x51

MPU6050_RA_EXT_SENS_DATA_09 0x52

MPU6050_RA_EXT_SENS_DATA_10 0x53

MPU6050_RA_EXT_SENS_DATA_11 0x54

MPU6050_RA_EXT_SENS_DATA_12 0x55

MPU6050_RA_EXT_SENS_DATA_13 0x56

MPU6050_RA_EXT_SENS_DATA_14 0x57

MPU6050_RA_EXT_SENS_DATA_15 0x58

MPU6050_RA_EXT_SENS_DATA_16 0x59

MPU6050_RA_EXT_SENS_DATA_17 0x5A

MPU6050_RA_EXT_SENS_DATA_18 0x5B

MPU6050_RA_EXT_SENS_DATA_19 0x5C

MPU6050_RA_EXT_SENS_DATA_20 0x5D

MPU6050_RA_EXT_SENS_DATA_21 0x5E

MPU6050_RA_EXT_SENS_DATA_22 0x5F

MPU6050_RA_EXT_SENS_DATA_23 0x60

MPU6050_RA_MOT_DETECT_STATUS 0x61

MPU6050_RA_I2C_SLV0_DO 0x63

MPU6050_RA_I2C_SLV1_DO 0x64

MPU6050_RA_I2C_SLV2_DO 0x65

MPU6050_RA_I2C_SLV3_DO 0x66

MPU6050_RA_I2C_MST_DELAY_CTRL 0x67

MPU6050_RA_SIGNAL_PATH_RESET 0x68

MPU6050_RA_MOT_DETECT_CTRL 0x69

MPU6050_RA_USER_CTRL 0x6A

MPU6050_RA_PWR_MGMT_1 0x6B

MPU6050_RA_PWR_MGMT_2 0x6C

MPU6050_RA_BANK_SEL 0x6D

MPU6050_RA_MEM_START_ADDR 0x6E

MPU6050_RA_MEM_R_W 0x6F

MPU6050_RA_DMP_CFG_1 0x70

MPU6050_RA_DMP_CFG_2 0x71

MPU6050_RA_FIFO_COUNTH 0x72

MPU6050_RA_FIFO_COUNTL 0x73

MPU6050_RA_FIFO_R_W 0x74

MPU6050_RA_WHO_AM_I 0x75

MPU6050_SELF_TEST_XA_1_BIT 0x07

MPU6050_SELF_TEST_XA_1_LENGTH 0x03

MPU6050_SELF_TEST_XA_2_BIT 0x05

MPU6050_SELF_TEST_XA_2_LENGTH 0x02

MPU6050_SELF_TEST_YA_1_BIT 0x07

MPU6050_SELF_TEST_YA_1_LENGTH 0x03

MPU6050_SELF_TEST_YA_2_BIT 0x03

MPU6050_SELF_TEST_YA_2_LENGTH 0x02

MPU6050_SELF_TEST_ZA_1_BIT 0x07

MPU6050_SELF_TEST_ZA_1_LENGTH 0x03

MPU6050_SELF_TEST_ZA_2_BIT 0x01

MPU6050_SELF_TEST_ZA_2_LENGTH 0x02

MPU6050_SELF_TEST_XG_1_BIT 0x04

MPU6050_SELF_TEST_XG_1_LENGTH 0x05

MPU6050_SELF_TEST_YG_1_BIT 0x04

MPU6050_SELF_TEST_YG_1_LENGTH 0x05

MPU6050_SELF_TEST_ZG_1_BIT 0x04

MPU6050_SELF_TEST_ZG_1_LENGTH 0x05

MPU6050_TC_PWR_MODE_BIT 7

MPU6050_TC_OFFSET_BIT 6

MPU6050_TC_OFFSET_LENGTH 6

MPU6050_TC_OTP_BNK_VLD_BIT 0

MPU6050_VDDIO_LEVEL_VLOGIC 0

MPU6050_VDDIO_LEVEL_VDD 1

MPU6050_CFG_EXT_SYNC_SET_BIT 5

MPU6050_CFG_EXT_SYNC_SET_LENGTH 3

MPU6050_CFG_DLPF_CFG_BIT 2

MPU6050_CFG_DLPF_CFG_LENGTH 3

MPU6050_EXT_SYNC_DISABLED 0x0

MPU6050_EXT_SYNC_TEMP_OUT_L 0x1

MPU6050_EXT_SYNC_GYRO_XOUT_L 0x2

MPU6050_EXT_SYNC_GYRO_YOUT_L 0x3

MPU6050_EXT_SYNC_GYRO_ZOUT_L 0x4

MPU6050_EXT_SYNC_ACCEL_XOUT_L 0x5

MPU6050_EXT_SYNC_ACCEL_YOUT_L 0x6

MPU6050_EXT_SYNC_ACCEL_ZOUT_L 0x7

MPU6050_DLPF_BW_256 0x00

MPU6050_DLPF_BW_188 0x01

MPU6050_DLPF_BW_98 0x02

MPU6050_DLPF_BW_42 0x03

MPU6050_DLPF_BW_20 0x04

MPU6050_DLPF_BW_10 0x05

MPU6050_DLPF_BW_5 0x06

MPU6050_GCONFIG_FS_SEL_BIT 4

MPU6050_GCONFIG_FS_SEL_LENGTH 2

MPU6050_GYRO_FS_250 0x00

MPU6050_GYRO_FS_500 0x01

MPU6050_GYRO_FS_1000 0x02

MPU6050_GYRO_FS_2000 0x03

MPU6050_ACONFIG_XA_ST_BIT 7

MPU6050_ACONFIG_YA_ST_BIT 6

MPU6050_ACONFIG_ZA_ST_BIT 5

MPU6050_ACONFIG_AFS_SEL_BIT 4

MPU6050_ACONFIG_AFS_SEL_LENGTH 2

MPU6050_ACONFIG_ACCEL_HPF_BIT 2

MPU6050_ACONFIG_ACCEL_HPF_LENGTH 3

MPU6050_ACCEL_FS_2 0x00

MPU6050_ACCEL_FS_4 0x01

MPU6050_ACCEL_FS_8 0x02

MPU6050_ACCEL_FS_16 0x03

MPU6050_DHPF_RESET 0x00

MPU6050_DHPF_5 0x01

MPU6050_DHPF_2P5 0x02

MPU6050_DHPF_1P25 0x03

MPU6050_DHPF_0P63 0x04

MPU6050_DHPF_HOLD 0x07

MPU6050_TEMP_FIFO_EN_BIT 7

MPU6050_XG_FIFO_EN_BIT 6

MPU6050_YG_FIFO_EN_BIT 5

MPU6050_ZG_FIFO_EN_BIT 4

MPU6050_ACCEL_FIFO_EN_BIT 3

MPU6050_SLV2_FIFO_EN_BIT 2

MPU6050_SLV1_FIFO_EN_BIT 1

MPU6050_SLV0_FIFO_EN_BIT 0

MPU6050_MULT_MST_EN_BIT 7

MPU6050_WAIT_FOR_ES_BIT 6

MPU6050_SLV_3_FIFO_EN_BIT 5

MPU6050_I2C_MST_P_NSR_BIT 4

MPU6050_I2C_MST_CLK_BIT 3

MPU6050_I2C_MST_CLK_LENGTH 4

MPU6050_CLOCK_DIV_348 0x0

MPU6050_CLOCK_DIV_333 0x1

MPU6050_CLOCK_DIV_320 0x2

MPU6050_CLOCK_DIV_308 0x3

MPU6050_CLOCK_DIV_296 0x4

MPU6050_CLOCK_DIV_286 0x5

MPU6050_CLOCK_DIV_276 0x6

MPU6050_CLOCK_DIV_267 0x7

MPU6050_CLOCK_DIV_258 0x8

MPU6050_CLOCK_DIV_500 0x9

MPU6050_CLOCK_DIV_471 0xA

MPU6050_CLOCK_DIV_444 0xB

MPU6050_CLOCK_DIV_421 0xC

MPU6050_CLOCK_DIV_400 0xD

MPU6050_CLOCK_DIV_381 0xE

MPU6050_CLOCK_DIV_364 0xF

MPU6050_I2C_SLV_RW_BIT 7

MPU6050_I2C_SLV_ADDR_BIT 6

MPU6050_I2C_SLV_ADDR_LENGTH 7

MPU6050_I2C_SLV_EN_BIT 7

MPU6050_I2C_SLV_BYTE_SW_BIT 6

MPU6050_I2C_SLV_REG_DIS_BIT 5

MPU6050_I2C_SLV_GRP_BIT 4

MPU6050_I2C_SLV_LEN_BIT 3

MPU6050_I2C_SLV_LEN_LENGTH 4

MPU6050_I2C_SLV4_RW_BIT 7

MPU6050_I2C_SLV4_ADDR_BIT 6

MPU6050_I2C_SLV4_ADDR_LENGTH 7

MPU6050_I2C_SLV4_EN_BIT 7

MPU6050_I2C_SLV4_INT_EN_BIT 6

MPU6050_I2C_SLV4_REG_DIS_BIT 5

MPU6050_I2C_SLV4_MST_DLY_BIT 4

MPU6050_I2C_SLV4_MST_DLY_LENGTH 5

MPU6050_MST_PASS_THROUGH_BIT 7

MPU6050_MST_I2C_SLV4_DONE_BIT 6

MPU6050_MST_I2C_LOST_ARB_BIT 5

MPU6050_MST_I2C_SLV4_NACK_BIT 4

MPU6050_MST_I2C_SLV3_NACK_BIT 3

MPU6050_MST_I2C_SLV2_NACK_BIT 2

MPU6050_MST_I2C_SLV1_NACK_BIT 1

MPU6050_MST_I2C_SLV0_NACK_BIT 0

MPU6050_INTCFG_INT_LEVEL_BIT 7

MPU6050_INTCFG_INT_OPEN_BIT 6

MPU6050_INTCFG_LATCH_INT_EN_BIT 5

MPU6050_INTCFG_INT_RD_CLEAR_BIT 4

MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT 3

MPU6050_INTCFG_FSYNC_INT_EN_BIT 2

MPU6050_INTCFG_I2C_BYPASS_EN_BIT 1

MPU6050_INTCFG_CLKOUT_EN_BIT 0

MPU6050_INTMODE_ACTIVEHIGH 0x00

MPU6050_INTMODE_ACTIVELOW 0x01

MPU6050_INTDRV_PUSHPULL 0x00

MPU6050_INTDRV_OPENDRAIN 0x01

MPU6050_INTLATCH_50USPULSE 0x00

MPU6050_INTLATCH_WAITCLEAR 0x01

MPU6050_INTCLEAR_STATUSREAD 0x00

MPU6050_INTCLEAR_ANYREAD 0x01

MPU6050_INTERRUPT_FF_BIT 7

MPU6050_INTERRUPT_MOT_BIT 6

MPU6050_INTERRUPT_ZMOT_BIT 5

MPU6050_INTERRUPT_FIFO_OFLOW_BIT 4

MPU6050_INTERRUPT_I2C_MST_INT_BIT 3

MPU6050_INTERRUPT_PLL_RDY_INT_BIT 2

MPU6050_INTERRUPT_DMP_INT_BIT 1

MPU6050_INTERRUPT_DATA_RDY_BIT 0

MPU6050_DMPINT_5_BIT 5

MPU6050_DMPINT_4_BIT 4

MPU6050_DMPINT_3_BIT 3

MPU6050_DMPINT_2_BIT 2

MPU6050_DMPINT_1_BIT 1

MPU6050_DMPINT_0_BIT 0

MPU6050_MOTION_MOT_XNEG_BIT 7

MPU6050_MOTION_MOT_XPOS_BIT 6

MPU6050_MOTION_MOT_YNEG_BIT 5

MPU6050_MOTION_MOT_YPOS_BIT 4

MPU6050_MOTION_MOT_ZNEG_BIT 3

MPU6050_MOTION_MOT_ZPOS_BIT 2

MPU6050_MOTION_MOT_ZRMOT_BIT 0

MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT 7

MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT 4

MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT 3

MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT 2

MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT 1

MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT 0

MPU6050_PATHRESET_GYRO_RESET_BIT 2

MPU6050_PATHRESET_ACCEL_RESET_BIT 1

MPU6050_PATHRESET_TEMP_RESET_BIT 0

MPU6050_DETECT_ACCEL_ON_DELAY_BIT 5

MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH 2

MPU6050_DETECT_FF_COUNT_BIT 3

MPU6050_DETECT_FF_COUNT_LENGTH 2

MPU6050_DETECT_MOT_COUNT_BIT 1

MPU6050_DETECT_MOT_COUNT_LENGTH 2

MPU6050_DETECT_DECREMENT_RESET 0x0

MPU6050_DETECT_DECREMENT_1 0x1

MPU6050_DETECT_DECREMENT_2 0x2

MPU6050_DETECT_DECREMENT_4 0x3

MPU6050_USERCTRL_DMP_EN_BIT 7

MPU6050_USERCTRL_FIFO_EN_BIT 6

MPU6050_USERCTRL_I2C_MST_EN_BIT 5

MPU6050_USERCTRL_I2C_IF_DIS_BIT 4

MPU6050_USERCTRL_DMP_RESET_BIT 3

MPU6050_USERCTRL_FIFO_RESET_BIT 2

MPU6050_USERCTRL_I2C_MST_RESET_BIT 1

MPU6050_USERCTRL_SIG_COND_RESET_BIT 0

MPU6050_PWR1_DEVICE_RESET_BIT 7

MPU6050_PWR1_SLEEP_BIT 6

MPU6050_PWR1_CYCLE_BIT 5

MPU6050_PWR1_TEMP_DIS_BIT 3

MPU6050_PWR1_CLKSEL_BIT 2

MPU6050_PWR1_CLKSEL_LENGTH 3

MPU6050_CLOCK_INTERNAL 0x00

MPU6050_CLOCK_PLL_XGYRO 0x01

MPU6050_CLOCK_PLL_YGYRO 0x02

MPU6050_CLOCK_PLL_ZGYRO 0x03

MPU6050_CLOCK_PLL_EXT32K 0x04

MPU6050_CLOCK_PLL_EXT19M 0x05

MPU6050_CLOCK_KEEP_RESET 0x07

MPU6050_PWR2_LP_WAKE_CTRL_BIT 7

MPU6050_PWR2_LP_WAKE_CTRL_LENGTH 2

MPU6050_PWR2_STBY_XA_BIT 5

MPU6050_PWR2_STBY_YA_BIT 4

MPU6050_PWR2_STBY_ZA_BIT 3

MPU6050_PWR2_STBY_XG_BIT 2

MPU6050_PWR2_STBY_YG_BIT 1

MPU6050_PWR2_STBY_ZG_BIT 0

MPU6050_WAKE_FREQ_1P25 0x0

MPU6050_WAKE_FREQ_2P5 0x1

MPU6050_WAKE_FREQ_5 0x2

MPU6050_WAKE_FREQ_10 0x3

MPU6050_BANKSEL_PRFTCH_EN_BIT 6

MPU6050_BANKSEL_CFG_USER_BANK_BIT 5

MPU6050_BANKSEL_MEM_SEL_BIT 4

MPU6050_BANKSEL_MEM_SEL_LENGTH 5

MPU6050_WHO_AM_I_BIT 6

MPU6050_WHO_AM_I_LENGTH 6

MPU6050_DMP_MEMORY_BANKS 8

MPU6050_DMP_MEMORY_BANK_SIZE 256

MPU6050_DMP_MEMORY_CHUNK_SIZE 16

file **MPU6050_6Axis_MotionApps20.h**
#include "I2Cdev.h"#include "helper_3dmath.h"#include "MPU6050.h"#include <inttypes.h>

Defines

MPU6050_INCLUDE_DMP_MOTIONAPPS20

__PGMSPACE_H_ 1

PROGMEM

PGM_P const char *

PSTR(str) (str)

F(x) x

strcpy_P(dest, src) strcpy((dest), (src))

strcat_P(dest, src) strcat((dest), (src))

strcmp_P(a, b) strcmp((a), (b))

pgm_read_byte(addr) (*(const unsigned char *)(addr))

pgm_read_word(addr) (*(const unsigned short *)(addr))

pgm_read_dword(addr) (*(const unsigned long *)(addr))

pgm_read_float(addr) (*(const float *)(addr))

pgm_read_byte_near(addr) pgm_read_byte(addr)

pgm_read_word_near(addr) pgm_read_word(addr)

pgm_read_dword_near(addr) pgm_read_dword(addr)

pgm_read_float_near(addr) pgm_read_float(addr)

pgm_read_byte_far(addr) pgm_read_byte(addr)

pgm_read_word_far(addr) pgm_read_word(addr)

pgm_read_dword_far(addr) pgm_read_dword(addr)

pgm_read_float_far(addr) pgm_read_float(addr)

DEBUG_PRINT(x)

DEBUG_PRINTF(x, y)

DEBUG_PRINTLN(x)

DEBUG_PRINTLNF(x, y)

MPU6050_DMP_CODE_SIZE 1929

MPU6050_DMP_CONFIG_SIZE 192

MPU6050_DMP_UPDATES_SIZE 47

Typedefs

typedef void **prog_void**

typedef char **prog_char**

typedef unsigned char **prog_uchar**

typedef int8_t **prog_int8_t**

typedef uint8_t **prog_uint8_t**

typedef int16_t **prog_int16_t**

typedef uint16_t **prog_uint16_t**

typedef int32_t **prog_int32_t**

typedef uint32_t **prog_uint32_t**

Variables

const unsigned char dmpUpdates [MPU6050_DMP_UPDATES_SIZE] PROGMEM

file **MPU6050_9Axis_MotionApps41.h**
#include "I2Cdev.h"#include "helper_3dmath.h"#include "MPU6050.h"#include <inttypes.h>

Defines

MPU6050_INCLUDE_DMP_MOTIONAPPS41

__PGMSPACE_H_ 1

PROGMEM

PGM_P const char *

PSTR(str) (str)

F(x) x

strcpy_P(dest, src) strcpy((dest), (src))

strcat_P(dest, src) strcat((dest), (src))

strcmp_P(a, b) strcmp((a), (b))

pgm_read_byte(addr) (*(const unsigned char *)(addr))

pgm_read_word(addr) (*(const unsigned short *)(addr))

pgm_read_dword(addr) (*(const unsigned long *)(addr))

pgm_read_float(addr) (*(const float *)(addr))

pgm_read_byte_near(addr) pgm_read_byte(addr)

pgm_read_word_near(addr) pgm_read_word(addr)

pgm_read_dword_near(addr) pgm_read_dword(addr)

pgm_read_float_near(addr) pgm_read_float(addr)

pgm_read_byte_far(addr) pgm_read_byte(addr)

pgm_read_word_far(addr) pgm_read_word(addr)

pgm_read_dword_far(addr) pgm_read_dword(addr)

pgm_read_float_far(addr) pgm_read_float(addr)

DEBUG_PRINT(x)

DEBUG_PRINTF(x, y)

DEBUG_PRINTLN(x)

DEBUG_PRINTLNF(x, y)

MPU6050_DMP_CODE_SIZE 1962

MPU6050_DMP_CONFIG_SIZE 232

MPU6050_DMP_UPDATES_SIZE 140

Typedefs

typedef void **prog_void**

typedef char **prog_char**

typedef int8_t **prog_int8_t**

typedef uint8_t **prog_uint8_t**

typedef int16_t **prog_int16_t**

typedef uint16_t **prog_uint16_t**

typedef int32_t **prog_int32_t**

typedef uint32_t **prog_uint32_t**

Variables

const unsigned char dmpUpdates [MPU6050_DMP_UPDATES_SIZE] PROGMEM

file **pb.h**

#include <stdint.h>#include <stddef.h>#include <stdbool.h>#include <string.h>

Defines

NANOPB_VERSION nanopb-0.3.8-dev

PB_PACKED_STRUCT_START

PB_PACKED_STRUCT_END

pb_packed

PB_UNUSED(x) (void)(x)

PB_STATIC_ASSERT(COND, MSG) typedef char PB_STATIC_ASSERT_MSG(MSG, __LINE__, __COUNTER__)[(COND)?1:-1];

PB_STATIC_ASSERT_MSG(MSG, LINE, COUNTER) PB_STATIC_ASSERT_MSG_(MSG, LINE, COUNTER)

PB_STATIC_ASSERT_MSG_(MSG, LINE, COUNTER) pb_static_assertion_##MSG##LINE##COUNTER

PB_MAX_REQUIRED_FIELDS 64

PB_LTYPE_VARINT 0x00 /* int32, int64, enum, bool */

PB_LTYPE_UVARINT 0x01 /* uint32, uint64 */

PB_LTYPE_SVARINT 0x02 /* sint32, sint64 */

PB_LTYPE_FIXED32 0x03 /* fixed32, sfixed32, float */

PB_LTYPE_FIXED64 0x04 /* fixed64, sfixed64, double */

PB_LTYPE_LAST_PACKABLE 0x04

PB_LTYPE_BYTES 0x05

PB_LTYPE_STRING 0x06

PB_LTYPE_SUBMESSAGE 0x07

PB_LTYPE_EXTENSION 0x08

PB_LTYPE_FIXED_LENGTH_BYTES 0x09

PB_LTYPES_COUNT 0x0A

PB_LTYPE_MASK 0x0F

PB_HTYPE_REQUIRED 0x00

PB_HTYPE_OPTIONAL 0x10

PB_HTYPE_REPEATED 0x20

PB_HTYPE_ONEOF 0x30

PB_HTYPE_MASK 0x30

PB_ATYPE_STATIC 0x00

PB_ATYPE_POINTER 0x80

PB_ATYPE_CALLBACK 0x40

PB_ATYPE_MASK 0xC0

PB_ATYPE(x) ((x) & PB_ATYPE_MASK)

PB_HTYPE(x) ((x) & PB_HTYPE_MASK)

PB_LTYPE(x) ((x) & PB_LTYPE_MASK)

PB_SIZE_MAX ((pb_size_t)-1)

PB_BYTES_ARRAY_T(n) struct { pb_size_t size; pb_byte_t bytes[n]; }

PB_BYTES_ARRAY_T_ALLOCSIZE(n) ((size_t)n + offsetof(*pb_bytes_array_t*, bytes))

PB_PROTO_HEADER_VERSION 30

pb_membersize(st, m) (sizeof ((st*)0)->m)

pb_arraysize(st, m) (pb_membersize(st, m) / pb_membersize(st, m[0]))

pb_delta(st, m1, m2) ((int)offsetof(st, m1) - (int)offsetof(st, m2))

PB_LAST_FIELD {0,(pb_type_t) 0,0,0,0,0,0}

PB_DATAOFFSET_FIRST(st, m1, m2) (offsetof(st, m1))

PB_DATAOFFSET_OTHER(st, m1, m2) (offsetof(st, m1) - offsetof(st, m2) - pb_membersize(st, m2))

PB_DATAOFFSET_CHOOSE(st, m1, m2) (int)(offsetof(st, m1) == offsetof(st, m2) \ ? PB_DATAOFFSET_FIRST(st, m1, m2) \ : PB_DATAOFFSET_OTHER(st, m1, m2))

PB_REQUIRED_STATIC(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_REQUIRED | ltype, \ fd, 0, pb_membersize(st, m), 0, ptr}

PB_OPTIONAL_STATIC(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_OPTIONAL | ltype, \ fd, \ pb_delta(st, has_ ## m, m), \ pb_membersize(st, m), 0, ptr}

PB_SINGULAR_STATIC(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_OPTIONAL | ltype, \ fd, 0, pb_membersize(st, m), 0, ptr}

PB_REPEATED_STATIC(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_REPEATED | ltype, \ fd, \ pb_delta(st, m ## _count, m), \ pb_membersize(st, m[0]), \ pb_arraysize(st, m), ptr}

PB_REQUIRED_INLINE(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_REQUIRED | PB_LTYPE_FIXED_LENGTH_BYTES, \ fd, 0, pb_membersize(st, m), 0, ptr}

PB_OPTIONAL_INLINE(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_OPTIONAL | PB_LTYPE_FIXED_LENGTH_BYTES, \ fd, \ pb_delta(st, has_ ## m, m), \ pb_membersize(st, m), 0, ptr}

PB_REQUIRED_POINTER(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_POINTER | PB_HTYPE_REQUIRED | ltype, \ fd, 0, pb_membersize(st, m[0]), 0, ptr}

PB_OPTIONAL_POINTER(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_POINTER | PB_HTYPE_OPTIONAL | ltype, \ fd, 0, pb_membersize(st, m[0]), 0, ptr}

PB_SINGULAR_POINTER(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_POINTER | PB_HTYPE_OPTIONAL | ltype, \ fd, 0, pb_membersize(st, m[0]), 0, ptr}

PB_REPEATED_POINTER(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_POINTER | PB_HTYPE_REPEATED | ltype, \ fd, pb_delta(st, m ## _count, m), \ pb_membersize(st, m[0]), 0, ptr}

PB_REQUIRED_CALLBACK(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_CALLBACK | PB_HTYPE_REQUIRED | ltype, \ fd, 0, pb_membersize(st, m), 0, ptr}

PB_OPTIONAL_CALLBACK(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_CALLBACK | PB_HTYPE_OPTIONAL | ltype, \ fd, 0, pb_membersize(st, m), 0, ptr}

PB_SINGULAR_CALLBACK(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_CALLBACK | PB_HTYPE_OPTIONAL | ltype, \ fd, 0, pb_membersize(st, m), 0, ptr}

PB_REPEATED_CALLBACK(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_CALLBACK | PB_HTYPE_REPEATED | ltype, \ fd, 0, pb_membersize(st, m), 0, ptr}

PB_OPTEXT_STATIC(tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_OPTIONAL | ltype, \ 0, \ 0, \ pb_membersize(st, m), 0, ptr}

PB_OPTEXT_POINTER(tag, st, m, fd, ltype, ptr) PB_OPTIONAL_POINTER(tag, st, m, fd, ltype, ptr)

PB_OPTEXT_CALLBACK(tag, st, m, fd, ltype, ptr) PB_OPTIONAL_CALLBACK(tag, st, m, fd, ltype, ptr)

PB_LTYPE_MAP_BOOL PB_LTYPE_VARINT

PB_LTYPE_MAP_BYTES PB_LTYPE_BYTES

PB_LTYPE_MAP_DOUBLE PB_LTYPE_FIXED64

PB_LTYPE_MAP_ENUM PB_LTYPE_VARINT

PB_LTYPE_MAP_UENUM PB_LTYPE_UVARINT

PB_LTYPE_MAP_FIXED32 PB_LTYPE_FIXED32

PB_LTYPE_MAP_FIXED64 PB_LTYPE_FIXED64

PB_LTYPE_MAP_FLOAT PB_LTYPE_FIXED32

PB_LTYPE_MAP_INT32 PB_LTYPE_VARINT

PB_LTYPE_MAP_INT64 PB_LTYPE_VARINT

PB_LTYPE_MAP_MESSAGE PB_LTYPE_SUBMESSAGE

PB_LTYPE_MAP_SFIXED32 PB_LTYPE_FIXED32

PB_LTYPE_MAP_SFIXED64 PB_LTYPE_FIXED64

PB_LTYPE_MAP_SINT32 PB_LTYPE_SVARINT

PB_LTYPE_MAP_SINT64 PB_LTYPE_SVARINT

PB_LTYPE_MAP_STRING PB_LTYPE_STRING

PB_LTYPE_MAP_UINT32 PB_LTYPE_UVARINT

PB_LTYPE_MAP_UINT64 PB_LTYPE_UVARINT

PB_LTYPE_MAP_EXTENSION PB_LTYPE_EXTENSION

PB_FIELD(tag, type, rules, allocation, placement, message, field, prevfield, ptr) PB_ ## rules ## _ ## allocation(tag, message, field, \ PB_DATAOFFSET_ ## placement(message, field, prevfield), \ PB_LTYPE_MAP_ ## type, ptr)

PB_ONEOF_STATIC(u, tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_ONEOF | ltype, \ fd, pb_delta(st, which_ ## u, u.m), \ pb_membersize(st, u.m), 0, ptr}

PB_ONEOF_POINTER(u, tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_POINTER | PB_HTYPE_ONEOF | ltype, \ fd, pb_delta(st, which_ ## u, u.m), \ pb_membersize(st, u.m[0]), 0, ptr}

PB_ONEOF_FIELD(union_name, tag, type, rules, allocation, placement, message, field, prevfield, ptr) PB_ONEOF_ ## allocation(union_name, tag, message, field, \ PB_DATAOFFSET_ ## placement(message, union_name.field, prevfield), \ PB_LTYPE_MAP_ ## type, ptr)

PB_ANONYMOUS_ONEOF_STATIC(u, tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_STATIC | PB_HTYPE_ONEOF | ltype, \ fd, pb_delta(st, which_ ## u, m), \ pb_membersize(st, m), 0, ptr}

PB_ANONYMOUS_ONEOF_POINTER(u, tag, st, m, fd, ltype, ptr) {tag, PB_ATYPE_POINTER | PB_HTYPE_ONEOF | ltype, \ fd, pb_delta(st, which_ ## u, m), \ pb_membersize(st, m[0]), 0, ptr}

PB_ANONYMOUS_ONEOF_FIELD(union_name, tag, type, rules, allocation, placement, message, field, prevfield, ptr) PB_ANONYMOUS_ONEOF_ ## allocation(union_name, tag, message, field, \ PB_DATAOFFSET_ ## placement(message, field, prevfield), \ PB_LTYPE_MAP_ ## type, ptr)

PB_SET_ERROR(stream, msg) (stream->errmsg = (stream)->errmsg ? (stream)->errmsg : (msg))

PB_GET_ERROR(stream) ((stream)->errmsg ? (stream)->errmsg : "(none)")

PB_RETURN_ERROR(stream, msg) return PB_SET_ERROR(stream, msg), false

Typedefs

typedef uint_least8_t **pb_type_t**

typedef uint_least8_t **pb_size_t**

typedef int_least8_t **pb_ssize_t**

typedef uint_least8_t **pb_byte_t**

typedef typedefPB_PACKED_STRUCT_START struct pb_field_s pb_field_t

typedef struct *pb_bytes_array_s* **pb_bytes_array_t**

typedef struct *pb_istream_s* **pb_istream_t**

typedef struct *pb_ostream_s* **pb_ostream_t**

typedef struct *pb_callback_s* **pb_callback_t**

typedef struct *pb_extension_type_s* **pb_extension_type_t**

typedef struct *pb_extension_s* **pb_extension_t**

Enums

enum pb_wire_type_t

Values:

PB_WT_VARINT = 0

PB_WT_64BIT = 1

PB_WT_STRING = 2

PB_WT_32BIT = 5

Functions

PB_PACKED_STRUCT_END PB_STATIC_ASSERT(sizeof(int64_t) = =2 *sizeof(int32_t), INT64_T_W

Variables

struct *pb_field_s* **pb_packed**

file **pb_common.c**

#include "pb_common.h"

Functions

bool **pb_field_iter_begin** (*pb_field_iter_t* *iter, **const** pb_field_t *fields, void *dest_struct)

bool **pb_field_iter_next** (*pb_field_iter_t* *iter)

bool **pb_field_iter_find** (*pb_field_iter_t* *iter, uint32_t tag)

file **pb_common.h**
#include "pb.h"

Typedefs

typedef struct *pb_field_iter_s* **pb_field_iter_t**

Functions

bool **pb_field_iter_begin** (*pb_field_iter_t* *iter, **const** pb_field_t *fields, void *dest_struct)

bool **pb_field_iter_next** (*pb_field_iter_t* *iter)

bool **pb_field_iter_find** (*pb_field_iter_t* *iter, uint32_t tag)

file **pb_decode.c**
#include "pb.h"#include "pb_decode.h"#include "pb_common.h"

Defines

checkreturn

Typedefs

typedef bool(* pb_decoder_t) (pb_istream_t *stream, const pb_field_t *field, void *dest)

Functions

static bool checkreturn buf_read(pb_istream_t * stream, pb_byte_t * buf, size_t count

static bool checkreturn pb_decode_varint32(pb_istream_t * stream, uint32_t * dest)

static bool checkreturn read_raw_value(pb_istream_t * stream, pb_wire_type_t wire_typ

static bool checkreturn decode_static_field(pb_istream_t * stream, pb_wire_type_t wir

static bool checkreturn decode_callback_field(pb_istream_t * stream, pb_wire_type_t w

static bool checkreturn decode_field(pb_istream_t * stream, pb_wire_type_t wire_type,

static void **iter_from_extension** (*pb_field_iter_t* *iter, *pb_extension_t* *extension)

static bool checkreturn default_extension_decoder(pb_istream_t * stream, pb_extension

static bool checkreturn decode_extension(pb_istream_t * stream, uint32_t tag, pb_wire

static bool checkreturn find_extension_field(pb_field_iter_t * iter)

static void **pb_field_set_to_default** (*pb_field_iter_t* *iter)

static void **pb_message_set_to_defaults** (**const** pb_field_t *fields*[], void **dest_struct*)

static bool checkreturn pb_dec_varint(pb_istream_t * stream, const pb_field_t * field

static bool checkreturn pb_dec_uvarint(pb_istream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_dec_svarint(pb_istream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_dec_fixed32(pb_istream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_dec_fixed64(pb_istream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_dec_bytes(pb_istream_t * stream, const pb_field_t * field,

static bool checkreturn pb_dec_string(pb_istream_t * stream, const pb_field_t * field

static bool checkreturn pb_dec_submessage(pb_istream_t * stream, const pb_field_t * f

bool checkreturn pb_skip_varint(pb_istream_t * stream)

bool checkreturn pb_skip_string(pb_istream_t * stream)

bool checkreturn pb_read(pb_istream_t * stream, pb_byte_t * buf, size_t count)

static bool checkreturn pb_readbyte(pb_istream_t * stream, pb_byte_t * buf)

pb_istream_t **pb_istream_from_buffer** (**const** *pb_byte_t *buf*, size_t *bufsize*)

bool checkreturn pb_decode_varint(pb_istream_t * stream, uint64_t * dest)

bool checkreturn pb_decode_tag(pb_istream_t * stream, pb_wire_type_t * wire_type, uin

bool checkreturn pb_skip_field(pb_istream_t * stream, pb_wire_type_t wire_type)

bool checkreturn pb_make_string_substream(pb_istream_t * stream, pb_istream_t * subs

void **pb_close_string_substream** (*pb_istream_t *stream*, *pb_istream_t *substream*)

static bool checkreturn decode_pointer_field(pb_istream_t * stream, pb_wire_type_t wi

bool checkreturn pb_decode_noinit(pb_istream_t * stream, const pb_field_t fields[], v

bool checkreturn pb_decode(pb_istream_t * stream, const pb_field_t fields[], void * d

bool **pb_decode_delimited** (*pb_istream_t *stream*, **const** pb_field_t *fields*[], void **dest_struct*)

bool **pb_decode_svarint** (*pb_istream_t *stream*, int64_t **dest*)

bool **pb_decode_fixed32** (*pb_istream_t *stream*, void **dest*)

bool **pb_decode_fixed64** (*pb_istream_t *stream*, void **dest*)

Variables

&pb_dec_bytes, &pb_dec_string, &pb_dec_submessage, NULL, &pb_dec_bytes }]

file **pb_decode.h**

#include "pb.h"

Functions

bool **pb_decode** (*pb_istream_t *stream*, **const** pb_field_t *fields*[], void **dest_struct*)

bool **pb_decode_noinit** (*pb_istream_t *stream*, **const** pb_field_t *fields*[], void **dest_struct*)

bool **pb_decode_delimited** (*pb_istream_t *stream*, **const** pb_field_t *fields*[], void **dest_struct*)

pb_istream_t **pb_istream_from_buffer** (**const** *pb_byte_t* *buf, size_t *bufsize*)

bool **pb_read** (*pb_istream_t* *stream, *pb_byte_t* *buf, size_t *count*)

bool **pb_decode_tag** (*pb_istream_t* *stream, *pb_wire_type_t* *wire_type, uint32_t *tag, bool *eof*)

bool **pb_skip_field** (*pb_istream_t* *stream, *pb_wire_type_t* *wire_type*)

bool **pb_decode_varint** (*pb_istream_t* *stream, uint64_t *dest*)

bool **pb_decode_svarint** (*pb_istream_t* *stream, int64_t *dest*)

bool **pb_decode_fixed32** (*pb_istream_t* *stream, void *dest*)

bool **pb_decode_fixed64** (*pb_istream_t* *stream, void *dest*)

bool **pb_make_string_substream** (*pb_istream_t* *stream, *pb_istream_t* *substream*)

void **pb_close_string_substream** (*pb_istream_t* *stream, *pb_istream_t* *substream*)

file **pb_encode.c**

#include "pb.h"#include "pb_encode.h"#include "pb_common.h"

Defines

checkreturn

Typedefs

**typedef bool(* pb_encoder_t) (pb_ostream_t *stream, const pb_field_t *field, const void *

Functions

static bool checkreturn buf_write(pb_ostream_t * stream, const pb_byte_t * buf, size_t

static bool checkreturn encode_array(pb_ostream_t * stream, const pb_field_t * field,

static bool checkreturn encode_field(pb_ostream_t * stream, const pb_field_t * field,

static bool checkreturn default_extension_encoder(pb_ostream_t * stream, const pb_ext

static bool checkreturn encode_extension_field(pb_ostream_t * stream, const pb_field_t

static bool checkreturn pb_enc_varint(pb_ostream_t * stream, const pb_field_t * field

static bool checkreturn pb_enc_uvarint(pb_ostream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_enc_svarint(pb_ostream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_enc_fixed32(pb_ostream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_enc_fixed64(pb_ostream_t * stream, const pb_field_t * fiel

static bool checkreturn pb_enc_bytes(pb_ostream_t * stream, const pb_field_t * field,

static bool checkreturn pb_enc_string(pb_ostream_t * stream, const pb_field_t * field

static bool checkreturn pb_enc_submessage(pb_ostream_t * stream, const pb_field_t * f

pb_ostream_t **pb_ostream_from_buffer** (*pb_byte_t* *buf, size_t *bufsize*)

bool checkreturn pb_write(pb_ostream_t * stream, const pb_byte_t * buf, size_t count)

static bool checkreturn encode_basic_field(pb_ostream_t * stream, const pb_field_t *

static bool checkreturn encode_callback_field(pb_ostream_t * stream, const pb_field_t

static void ***remove_const** (**const** void **p*)

bool checkreturn pb_encode(pb_ostream_t * stream, const pb_field_t fields[], const vo

bool **pb_encode_delimited** (*pb_ostream_t* **stream*, **const** pb_field_t *fields*[], **const** void **src_struct*)

bool **pb_get_encoded_size** (size_t **size*, **const** pb_field_t *fields*[], **const** void **src_struct*)

bool checkreturn pb_encode_varint(pb_ostream_t * stream, uint64_t value)

bool checkreturn pb_encode_svarint(pb_ostream_t * stream, int64_t value)

bool checkreturn pb_encode_fixed32(pb_ostream_t * stream, const void * value)

bool checkreturn pb_encode_fixed64(pb_ostream_t * stream, const void * value)

bool checkreturn pb_encode_tag(pb_ostream_t * stream, pb_wire_type_t wiretype, uint32_

bool checkreturn pb_encode_tag_for_field(pb_ostream_t * stream, const pb_field_t * fi

bool checkreturn pb_encode_string(pb_ostream_t * stream, const pb_byte_t * buffer, si

bool checkreturn pb_encode_submessage(pb_ostream_t * stream, const pb_field_t fields[

Variables

&pb_enc_bytes, &pb_enc_string, &pb_enc_submessage, NULL, &pb_enc_bytes }]

file **pb_encode.h**

#include "pb.h"

Defines

PB_OSTREAM_SIZING {0,0,0,0,0}

Functions

bool **pb_encode** (*pb_ostream_t* **stream*, **const** pb_field_t *fields*[], **const** void **src_struct*)

bool **pb_encode_delimited** (*pb_ostream_t* **stream*, **const** pb_field_t *fields*[], **const** void **src_struct*)

bool **pb_get_encoded_size** (size_t **size*, **const** pb_field_t *fields*[], **const** void **src_struct*)

pb_ostream_t **pb_ostream_from_buffer** (*pb_byte_t* **buf*, size_t *bufsize*)

bool **pb_write** (*pb_ostream_t* **stream*, **const** *pb_byte_t* **buf*, size_t *count*)

bool **pb_encode_tag_for_field** (*pb_ostream_t* **stream*, **const** pb_field_t **field*)

bool **pb_encode_tag** (*pb_ostream_t* **stream*, *pb_wire_type_t* *wiretype*, uint32_t *field_number*)

bool **pb_encode_varint** (*pb_ostream_t* **stream*, uint64_t *value*)

bool **pb_encode_svarint** (*pb_ostream_t* **stream*, int64_t *value*)

bool **pb_encode_string** (*pb_ostream_t* **stream*, **const** *pb_byte_t* **buffer*, size_t *size*)

bool **pb_encode_fixed32** (*pb_ostream_t* **stream*, **const** void **value*)

bool **pb_encode_fixed64** (*pb_ostream_t* *stream, **const** void *value)

bool **pb_encode_submessage** (*pb_ostream_t* *stream, **const** pb_field_t fields[], **const** void *src_struct)

file **protoLighthouse.cpp**
#include "protoLighthouse.h"

file **protoLighthouse.h**
#include "logging.h"#include "pb.h"#include "lighthouse.pb.h"#include "pb_encode.h"#include "pb_decode.h"#include "helper_3dmath.h"

Typedefs

typedef enum *_ES* **ES_PROTO**

Enums

enum _ES
Values:

ES_WIFI_FAIL_INIT_NO_SHIELD = 1

ES_WIFI_FAIL_INIT_CANNOT_CONNECT

ES_WIFI_FAIL_UDP_SOCKET

ES_PROTO_FAIL_INIT

ES_PROTO_FAIL_ENCODE

ES_PROTO_FAIL_DECODE

ES_PROTO_ERROR

ES_PROTO_SUCCESS

ES_WIFI_SUCCESS

ES_WIFI_ERROR

file **SPISlave.cpp**
#include "SPISlave.h"#include "hspi_slave.h"

Variables

SPISlaveClass **SPISlave**

file **SPISlave.h**
#include "Arduino.h"#include <functional>

Typedefs

typedef *std*::function<void (uint8_t *data, size_t len)> **SpiSlaveDataHandler**

typedef *std*::function<void (uint32_t status)> **SpiSlaveStatusHandler**

typedef *std*::function<void (void)> **SpiSlaveSentHandler**

Variables

SPISlaveClass **SPISlave**

file **wirelessLove.cpp**
#include <SPI.h>#include "wirelessLove.h"

Variables

const char **TestBuffer**[] ="Hello World"

const int **timestampSize** = 2

file **wirelessLove.h**
#include <ESP8266WiFi.h>#include <WiFiClientSecure.h>#include <ESP8266WiFiGeneric.h>#include <ESP8266WiFiAP.h>#include <WiFiServer.h>#include <ESP8266WiFiScan.h>#include <ESP8266WiFiType.h>#include <ESP8266WiFiMulti.h>#include <ESP8266WiFiSTA.h>#include <WiFiClient.h>#include <WiFiUdp.h>#include "logging.h"#include "protoLighthouse.h"

Defines

TRIGGER_PIN 10

file **ConcatenateSensorValues.v**

file **Counter.v**

file **Counter.v**

file **DarkRoomControl.v**

file **pll_altpll.v**

file **fifo.v**

file **fifo_bb.v**

file **mg4v9.v**

file **mg4v9.v**

file **mg8t9.v**

file **mge0a.v**

file **mgsu9.v**

file **mgsu9.v**

file **lpm_mux0.v**

file **lpm_mux0_bb.v**

file **dff.v**

file **SpiControl.v**

file **SpiControl.v**

file **tb_Counter.v**

file **tb_dff.v**

file **tb_SpiControl.v**

file **oneshot.v**

file **mgbt9.v**

file **pll.v**

file **pll_bb.v**

file **sensor_select.v**

file **simple_counter.v**

file **spi_bb.v**

file **spi_inst.v**

file **spi.v**

file **altera_avalon_sc_fifo.v**

file **altera_reset_controller.v**

file **altera_reset_synchronizer.v**

file **spi_bridge_0.v**

file **spi_mm_interconnect_0.v**

file **spi_mm_interconnect_0_avalon_st_adapter.v**

file **spi_spi_0.v**

file **spi_vic_0.v**

file **DarkRoom.hpp**

#include <QPainter>#include <QCheckBox>#include <QPushButton>#include <QLineEdit>#include <QVBoxLayout>#include <QHBoxLayout>#include <QLabel>#include <QTableWidget>#include <QComboBox>#include <QTimer>#include <QScrollArea>#include <QListWidget>#include <QStyledItemDelegate>#include "darkroom/TrackedObjectDelegate.hpp"#include <map>#include <thread>#include <mutex>#include <ros/ros.h>#include <rviz/panel.h>#include <pluginlib/class_loader.h>#include <pluginlib/class_list_macros.h>#include <visualization_msgs/Marker.h>#include <std_srvs/SetBool.h>#include <tf/tf.h>#include <tf/transform_listener.h>#include <tf_conversions/tf_eigen.h>#include <Eigen/Core>#include <Eigen/Dense>#include "darkroom/TrackedObject.hpp"#include "darkroom/UDPSocket.hpp"

file **PoseMinimizer.hpp**

#include <tf/tf.h>#include <Eigen/Core>#include <Eigen/Dense>#include <unsupported/Eigen/NonLinearOptimization>#include <unsupported/Eigen/NumericalDiff>#include <iostream>

file **Sensor.hpp**

#include <ros/ros.h>#include <Eigen/Core>#include <Eigen/Dense>#include <Eigen/Geometry>#include <mutex>#include <vector>

Defines

NUMBER_OF_LIGHTHOUSES 2

Typedefs

typedef Matrix<unsigned short, 2, 1> **Vector2s**

file **TrackedObject.hpp**

#include <ros/ros.h>#include <visualization_msgs/Marker.h>#include <tf/tf.h>#include <tf/transform_listener.h>#include <tf/transform_broadcaster.h>#include <tf_conversions/tf_eigen.h>#include <Eigen/Core>#include <Eigen/Dense>#include <Eigen/Geometry>#include <unsupported/Eigen/NonLinearOptimization>#include <unsupported/Eigen/NumericalDiff>#include "base3d/triangulation.h"#include "base3d/projection.h"#include <map>#include <fstream>#include <deque>#include <thread>#include <mutex>#include "yaml-cpp/yaml.h"#include "darkroom/UDPSocket.hpp"#include "darkroom/Sensor.hpp"#include "darkroom/PoseMinimizer.hpp"

Defines

degreesToRadians(angleDegrees) (angleDegrees * M_PI / 180.0)

radiansToDegrees(angleRadians) (angleRadians * 180.0 / M_PI)

uSecsToRadians(ticks) (degreesToRadians(ticks * 0.0216))

MAX_ITERATIONS 30

DEFAULT_CONFIG_PORT 8001

DEFAULT_SENSOR_PORT 8002

DEFAULT_IMU_PORT 8003

DEFAULT_COMMAND_PORT 8004

DEFAULT_LOGGING_PORT 8005

Typedefs

typedef boost::shared_ptr<*TrackedObject*> **TrackedObjectPtr**

file **TrackedObjectDelegate.hpp**

#include <QWidget>#include <QStyledItemDelegate>#include <QMetaType>#include <QPointF>#include <QVector>#include <QPainter>#include <QMouseEvent>#include <QTableWidget>

Functions

void **populateTableWidget**(QTableWidget **tableWidget*)

file **UDPSocket.hpp**

#include <ros/ros.h>#include <stdlib.h>#include <unistd.h>#include <errno.h>#include <string.h>#include <sys/types.h>#include <sys/socket.h>#include <netinet/in.h>#include <arpa/inet.h>#include <netdb.h>#include <string>#include <ifaddrs.h>#include <stdio.h>#include <time.h>#include <bitset>#include "lighthouse.pb.h"

Defines

MAXBUFLENGTH 1024

BYTE_TO_BINARY_PATTERN "%c%c%c%c%c%c%c%c"

BYTE_TO_BINARY(byte) (byte & 0x80 ? '1' : '0'), \ (byte & 0x40 ? '1' : '0'), \ (byte & 0x20 ? '1' : '0'), \ (byte & 0x10 ? '1' : '0'), \ (byte & 0x08 ? '1' : '0'), \ (byte & 0x04 ? '1' : '0'), \ (byte & 0x02 ? '1' : '0'), \ (byte & 0x01 ? '1' : '0')

Typedefs

typedef boost::shared_ptr<*UDPSocket*> **UDPSocketPtr**

Enums

enum COMMAND

Values:

TRACKING = 1

MPU = 2

SENSOR = 3

RESET = 4

Functions

bool **convertByte2Text** (uint32_t *inet*, char **inet_str*)

bool **convertText2Byte** (char **inet_str*, uint32_t &*inet*)

file **CommunicationTest.cpp**

#include "darkroom/UDPSocket.hpp"

Functions

int **main** (int *argc*, char **argv*[])

file **DarkRoom.cpp**

#include "include/darkroom/DarkRoom.hpp"

file **PoseMinimizer.cpp**

#include "darkroom/PoseMinimizer.hpp"

file **Sensor.cpp**

#include "darkroom/Sensor.hpp"

file **TrackedObject.cpp**

#include "darkroom/TrackedObject.hpp"

file **TrackedObjectDelegate.cpp**

#include "darkroom/TrackedObjectDelegate.hpp"

Functions

void **populateTableWidget** (QTableWidget **tableWidget*)

Variables

const int **PaintingScaleFactor** = 20

file **UDPSocket.cpp**

#include "darkroom/UDPSocket.hpp"

Functions

bool **convertByte2Text** (uint32_t *inet*, char **inet_str*)

bool **convertText2Byte** (char **inet_str*, uint32_t **inet*)

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/include/d**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/include**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/src**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

dir **/home/docs/checkouts/readthedocs.org/user_builds/darkroom/checkouts/latest/src/DarkRoom_**

Solution Strategy

Todo

Describe your solution strategy.

Contents.

A short summary and explanation of the fundamental solution ideas and strategies.

Motivation.

An architecture is often based upon some key solution ideas or strategies. These ideas should be familiar to everyone involved into the architecture.

Form.

Diagrams and / or text, as appropriate. Keep it short, i.e. 1 or 2 pages at most!

Test Strategy

Todo

If you have any tests describe:

- Where the tests are kept.
- How the tests are invoked.
- What you can/need to test manually.

System Tests

Integration Tests

Unit Tests

Building Block View

Overview

Todo

Inset a building block view:

- Static decomposition of the system into building blocks and the relationships thereof.
- Description of libraries and software used
-

We specify the system based on the blackbox view from *UML System Context* by now considering it a whitebox and identifying the next layer of blackboxes inside it. We re-iterate this zoom-in until specific granularity is reached - 2 levels should be enough.

Motivation.

This is the most important view, that must be part of each architecture documentation. In building construction this would be the floor plan.

Tool

- Create diagrams as below.

The white box view of the first level of your code. This is a white box view of your system as shown within the in Context in figure: *UML System Context*. External libraries and software are clearly marked.

Level 1 - Components

The highest level components

Todo

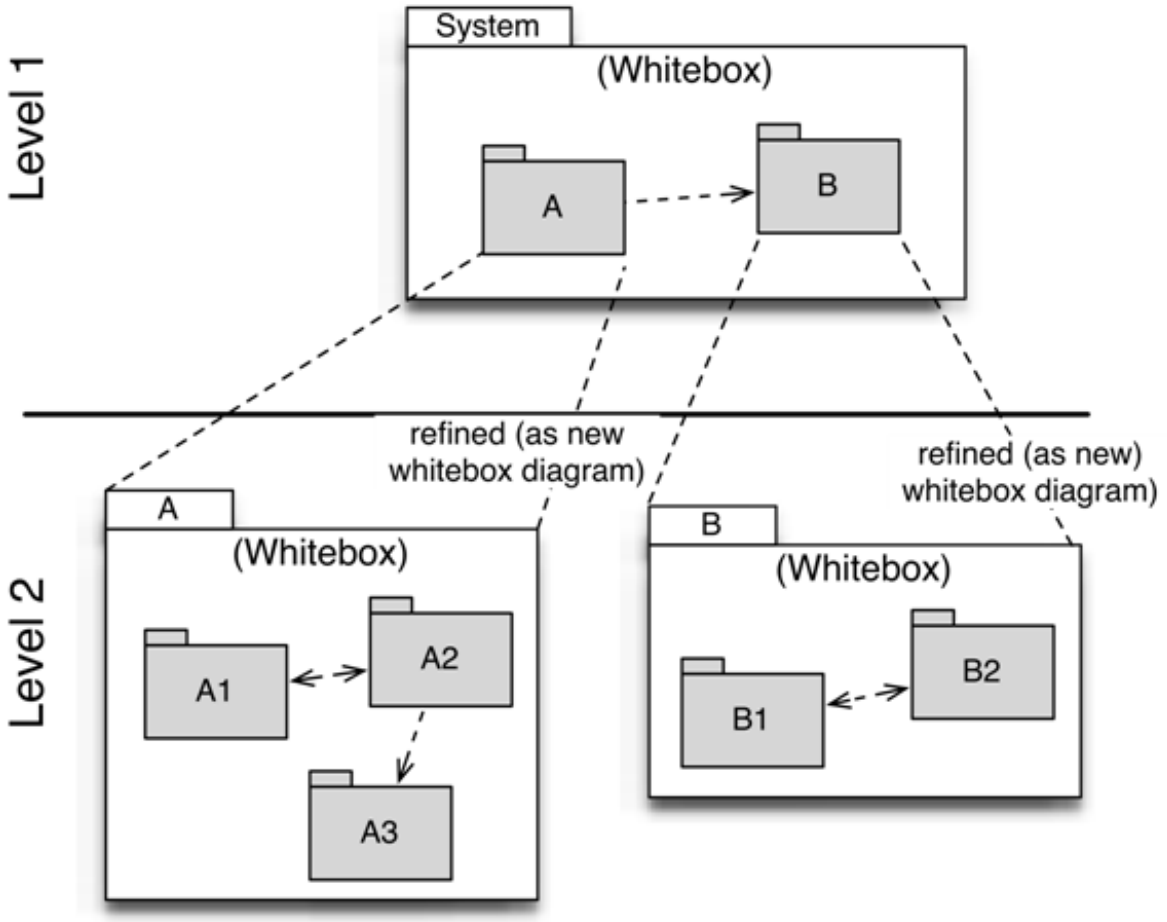

Fig. 1.4: Building blocks overview

- Define your groups using the **/defgroup** doxygen command
- Add **@addtogroup** tags to doxygen blocks of components in code as described here: http://www.stack.nl/~dimitri/doxygen/manual/grouping.html#modules
- Adapt the doxygencall to match the group name

Warning: doxygengroup: Cannot find namespace "nuttygroup" in doxygen xml output for project "DarkRoom" from directory: doxyxml/

Level 2 - Components within each component of level 1

Todo

- Define your groups using the **/defgroup** doxygen command
- Add **addtogroup** tags to doxygen blocks of components in code as described here: http://www.stack.nl/~dimitri/doxygen/manual/grouping.html#modules
- Adapt the doxygencall to match the group name

Warning: doxygengroup: Cannot find namespace "nuttygroup2" in doxygen xml output for project "DarkRoom" from directory: doxyxml/

Runtime View

Todo

Add a runtime view of i.e. the startup of your code. This should help people understand how your code operates and where to look if something is not working.

Contents. alternative terms: - Dynamic view - Process view - Workflow view This view describes the behavior and interaction of the system's building blocks as runtime elements (processes, tasks, activities, threads, ...).

Select interesting runtime scenarios such as:

- How are the most important use cases executed by the architectural building blocks?
- Which instances of architectural building blocks are created at runtime and how are they started, controlled, and stopped.
- How do the system's components co-operate with external and pre-existing components?
- How is the system started (covering e.g. required start scripts, dependencies on external systems, databases, communications systems, etc.)?

  Note

  The main criterion for the choice of possible scenarios (sequences, workflows) is their **architectural relevancy**. It is **not** important to describe a large number of scenarios. You should rather document a representative selection.

Candidates are:

1. The top 3 – 5 use cases
2. System startup
3. The system's behavior on its most important external interfaces
4. The system's behavior in the most important error situations

Motivation.

Especially for object-oriented architectures it is not sufficient to specify the building blocks with their interfaces, but also how instances of building blocks interact during runtime.

Form.

Document the chosen scenarios using UML sequence, activity or communications diagrams. Enumerated lists are sometimes feasible.

Using object diagrams you can depict snapshots of existing runtime objects as well as instantiated relationships. The UML allows to distinguish between active and passive objects.

Runtime Scenario 1

Runtime Scenario 2

...

Deployment View

Todo

Add a deployment diagram. **Contents.**

This view describes the environment within which the system is executed. It describes the geographic distribution of the system or the structure of the hardware components that execute the software. It documents workstations, processors, network topologies and channels, as well as other elements of the physical system environment.

Motivation.

Software is not much use without hardware. These models should enable the operator to properly install the software.

You can separate this into different levels...

Libraries and external Software

Contains a list of the libraries and external software used by this system.

Todo

List libraries you are using

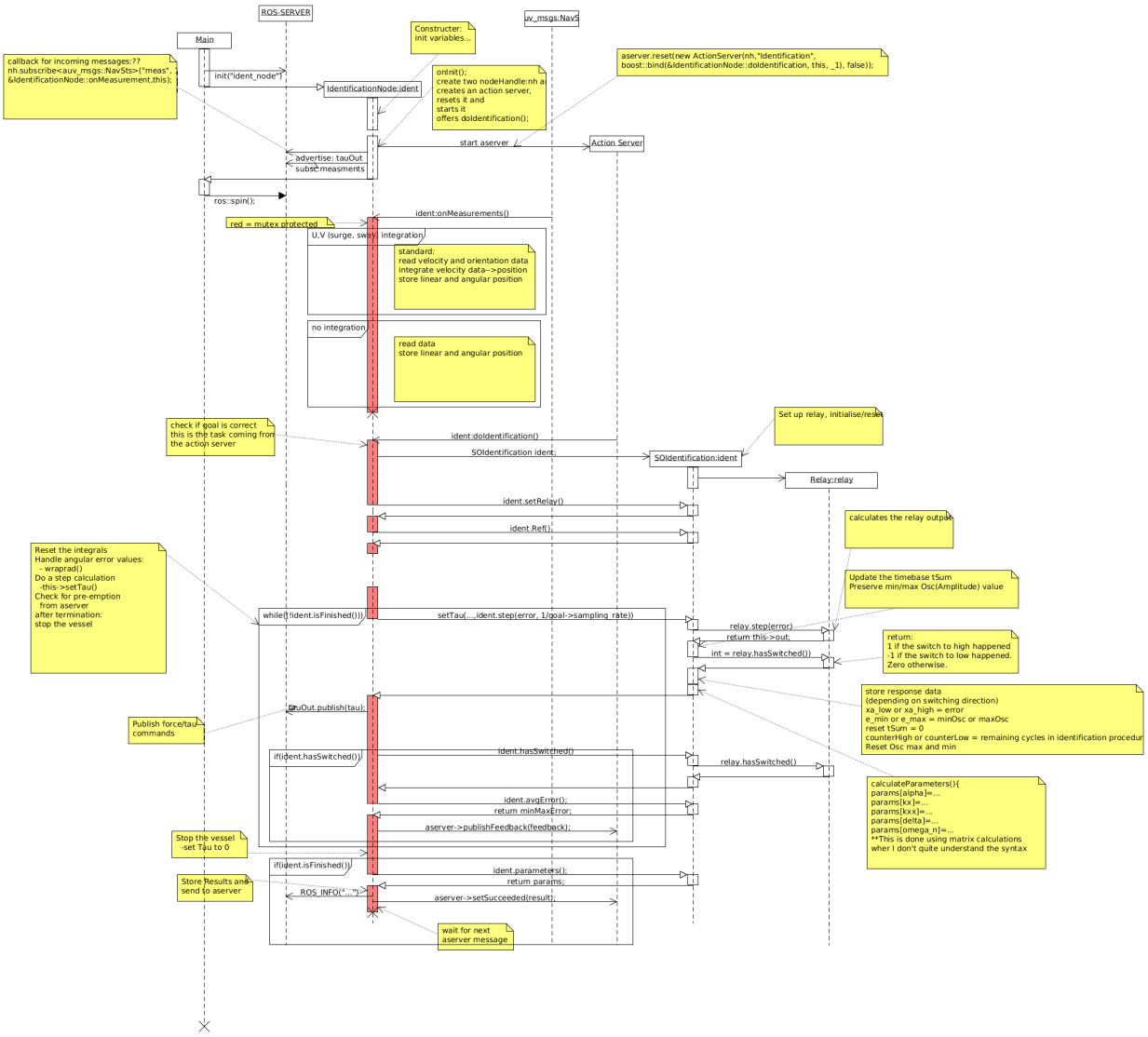

Fig. 1.5: **UML-type sequence diagram** - Shows how components interact with each other during runtime.

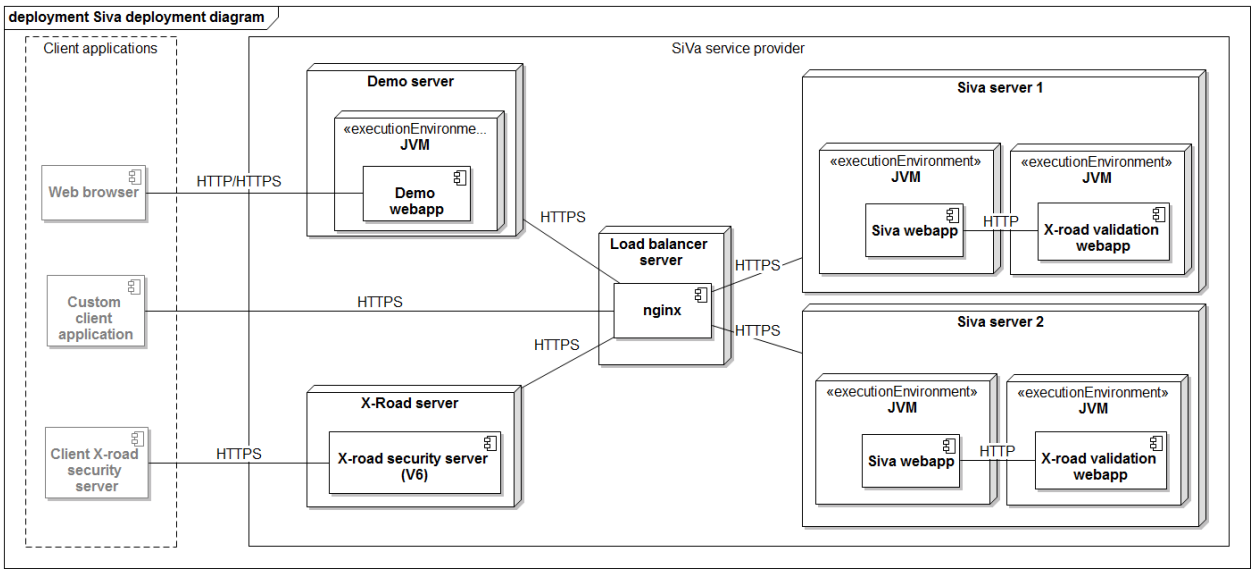

Table 1.6: Libraries and external Software

Name	URL/Author	License	Description
arc42	http://www.arc42.de/template/	Creative Commens Attribution license.	Template for documenting and developing software

About arc42

This information should stay in every repository as per their license: http://www.arc42.de/template/licence.html

arc42, the Template for documentation of software and system architecture.

By Dr. Gernot Starke, Dr. Peter Hruschka and contributors.

Template Revision: 6.5 EN (based on asciidoc), Juni 2014

© We acknowledge that this document uses material from the arc 42 architecture template, http://www.arc42.de. Created by Dr. Peter Hruschka & Dr. Gernot Starke. For additional contributors see http://arc42.de/sonstiges/contributors.html

Note

This version of the template contains some help and explanations. It is used for familiarization with arc42 and the understanding of the concepts. For documentation of your own system you use better the *plain* version.

Literature and references

Starke-2014 Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden. Carl Hanser Verlag, 6, Auflage 2014.

Starke-Hruschka-2011 Gernot Starke und Peter Hruschka: Softwarearchitektur kompakt. Springer Akademischer Verlag, 2. Auflage 2011.

Zörner-2013 Softwarearchitekturen dokumentieren und kommunizieren, Carl Hanser Verlag, 2012

Examples

- HTML Sanity Checker
- DocChess (german)
- Gradle (german)
- MaMa CRM (german)
- Financial Data Migration (german)

Acknowledgements and collaborations

arc42 originally envisioned by Dr. Peter Hruschka and Dr. Gernot Starke.

Sources We maintain arc42 in *asciidoc* format at the moment, hosted in GitHub under the aim42-Organisation.

Issues We maintain a list of open topics and bugs.

We are looking forward to your corrections and clarifications! Please fork the repository mentioned over this lines and send us a *pull request*!

Collaborators

We are very thankful and acknowledge the support and help provided by all active and former collaborators, uncountable (anonymous) advisors, bug finders and users of this method.

Currently active

- Gernot Starke
- Stefan Zörner
- Markus Schärtel
- Ralf D. Müller
- Peter Hruschka
- Jürgen Krey

Former collaborators

(in alphabetical order)

- Anne Aloysius
- Matthias Bohlen
- Karl Eilebrecht
- Manfred Ferken
- Phillip Ghadir
- Carsten Klein
- Prof. Arne Koschel
- Axel Scheithauer

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

P

U

V

W