SnAirBeats

1.0

Generated by Doxygen 1.13.2

Chapter 1

SnAirBeats

1.1 SnAIRbeats

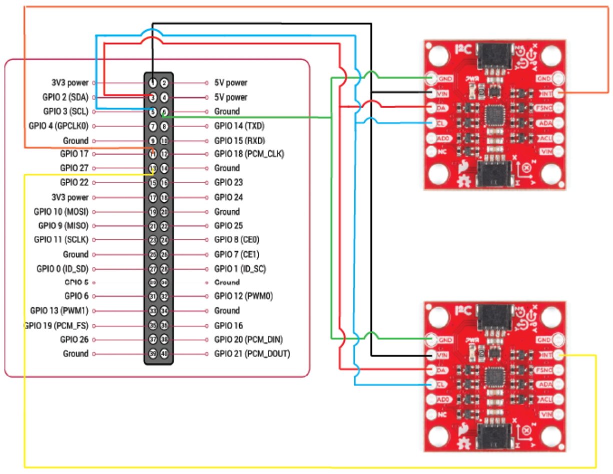
SnAirBeats is a next generation method to practice the drums, while reducing noise and space typically required to do so. The SnAirBeat set uses intertial measurement units (IMU) within the sticks to track their movement and play a corresponding drum, not requiring any physical hitting like modern electric drum sets need.

1.2 Building

SnAIRBeats requires the following components to work:

- 1x Raspberry Pi 5
- 2x SEN15335 Breakout IMU
- 1x External USB Speaker

The circuit's wires should be at least 1m long to ensure comfortable movement while playing to avoid risk of damaging the project. A wiring guide can be seen below:



The drumsticks for the project need to be 3D printed via the `STLs` provided within this repository.

1.3 Prerequisites

Firstly it should be noted that SnAIRBeats can only run on a Linux system. It is recommened to use a Raspberry Pi operating system such as `Raspebian` as the packages will not work on Windows systems.

Before installing any of the prerequisites, please update your package list with:

```
sudo apt update
```

There are 4 main libraries that need to be installed for this project:

- Libgpiod - for general purpose input/output
- mraa - IoT and hardware interface library (required for IMU driver)
- YAML - Support for YAML (required for IMU driver)
- ALSA - To process and play sound files

These packages can be installed by running the following commands through the terminal of the Raspberry Pi.

```
sudo apt install -y libgpiod-dev
sudo apt install -y libmraa-dev
sudo apt install -y libyaml-dev
sudo apt install -y libasound2-dev
```

1.4 Compliation from source

The project is built using a series of CMakeLists.txt which locate and link the required internal and external libraries for the project. By running the code below, the CMake will generate the respective make files within each of files. Running make will build the project and return an executable.

```
cmake .
make
```

It may take a few seconds for everything to build properly, but once everything has been successfully created you can use the code below to run SnAIRBeats.

```
./SnairBeats
```

1.5 Usage

SnAIRBeats works by reading the direction of acceleration within the IMUs. Holding the sticks with the X-direction representing the vertical axis:

- Hitting a stick down will play a snare drum
- Hitting a stick to either side will play a high tom
- Lunging the stick forward will play a crash cymbal

If desired, the sounds played by each direction can be changed by swapping files in the ALSAPlayer library found either `here` or through the command directory:

```
cd src/libs/ALSAPlayer/include
ls
```

1.5.1 Maximum Latency

The highest sampling rate the SEN 15335 IMUs can work at is 1.125kHz.
This value can be adjusted in the main.cpp file by altering the SampleRateDivider variable. This divides the sampling rate by 1+SampleRateDivider.

We have found that the maximum latency the sticks can be reliably played at is 25Hz (1125Hz/44+1). While decreasing the latency may improve the sensitivity of the sticks, the higher this value is the greater the power consumption will be.

1.6 Libraries

Here is a small description of each of the libraries used within the project and what they are used for.

1.6.1 ALSAPlayer

ALSAPlayer takes .wav files from inside its `include folder` and converts them into audio buffers using the ConvertFiles function. This library is heavily based off of driver written by `Adam Stark` found at `https://github.com/adamstark/AudioFile`.

Audio devices are opened using the Open function which once finished can be used to play the created audiobuffers using the playFile function. The playFile function is built to play small audios and will interrupt itself, cancelling whatever is playing to play the next audio. This is much easier for SnAIRBeats compared to mixing as the interrupt of the drum notes is not noticable to the human ear, especially with the sample delay between each hit.

1.6.2 GPIO

The GPIO library initialises the GPIO pins of the Raspberry Pi. Using `libgpiod`, an event driven interrupt function called "worker" is used to read one of the GPIO pins for a HIGH value. The function is blocked until a rising edge event is seen in the GPIO pin selected in the constructor.

The interrupt is data-ready based and therefore wakes whenever new data is available from the sensor. Within the constructor, 2 objects were passed in, the Maths object and the I2C-IMU driver. The new data is read from the IMU's registers using a read function and passed into a callback which inputs the data into the maths object to be thresholded.

1.6.3 I2C

The I2C library is a driver written specifically for the `ICM-20948 chip` seen within the SEN 15335 IMU and is very heavily based off of driver written by `NTKot` found at `https://github.com/NTkot/icm20948↩_i2c` with the Raw-Data-Ready interrupt turned on and the magnetometer turned off.

For each sensor used within the system, an object from this driver is built with a separate I2C address to differentiate between the two. These objects come with pre-built functions, must useful is the Read_Accel_Gyro which reads the registers of the IMU and stores the values in a variable within the object. These variables are what are passed into the IMUMaths callback through the GPIO worker whenever data is ready.

1.6.4 IMUMaths

This libary was written to threshold the data that came through from the GPIO worker and has two main goals. Firstly it reads the data passed through and checks whether any of the values correlate to a hit and then play the corresponding audio from the ALSAAudio object. It also contains a sample delay to stop multiple sounds being played from the same hit. This is achieved using a simple boolean that is turned true after a hit is detected and waits a set number of samples before the boolean flips back, allowing another hit to be detected.

1.7 Unit tests

This project uses unit testing to validate the functionality of the key classes, including classes responsible for IMU data processing and audio playback.

Tests are written using the GoogleTest framework and integrated with CTest for easy execution.

To run the tests from the root directory, use:

```
./run_tests
```

or to use CMake directly, run:

```
ctest
```

1.8 Documentation

Complete documentation for this project can be found in `documentation.pdf`.

1.9 Sponsorship and funding

We are very grateful for RS Components for providing us with components that allowed us to complete this project.

1.10 Media

- Instagram

1.11 Authors

- Calum Robertson
- Aleksandar Zahariev
- Mohammed Alqabandi
- Renata Cia Sanches Loberto
- Alejandra Paja Garcia

1.12 Licenses

The IMU driver has been adapted from the driver written by `NTKot` and can be found at `https://github.com/NTkot/icm20948_i2c`

The ALSAPlayer library has been adapted from the driver written by `Adam Stark` and can be found at `https://github.com/adamstark/AudioFile`

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 AudioPlayerName Namespace Reference

Classes

- class AudioPlayer

6.2 GPIOName Namespace Reference

Classes

- class GPIOClass
- struct MathsCallbackStruct

6.3 icm20948 Namespace Reference

Classes

- class ICM20948_I2C

6.4 IMUMathsName Namespace Reference

Classes

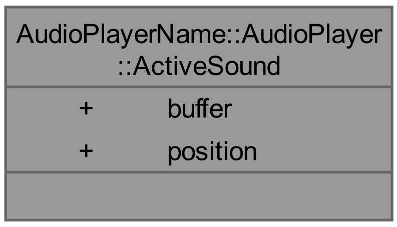
- struct AudioCallback
- class IMUMaths

Chapter 7

Class Documentation

7.1 AudioPlayerName::AudioPlayer::ActiveSound Struct Reference

Collaboration diagram for AudioPlayerName::AudioPlayer::ActiveSound:



Public Attributes

- std::vector< int32_t > ∗ buffer
- size_t position

7.1.1 Member Data Documentation

7.1.1.1 buffer

```
std::vector<int32_t>* AudioPlayerName::AudioPlayer::ActiveSound::buffer
```

7.1.1.2 position

```
size_t AudioPlayerName::AudioPlayer::ActiveSound::position
```

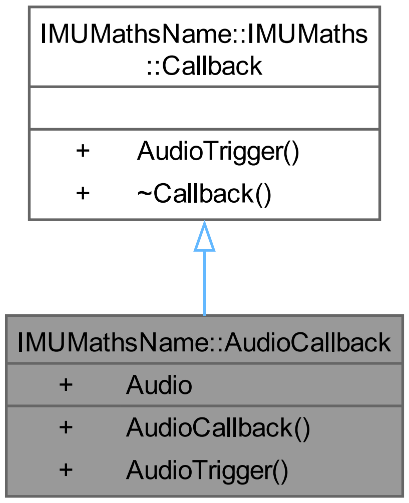
The documentation for this struct was generated from the following file:

- src/libs/ALSAPlayer/include/ALSAPlayer.hpp

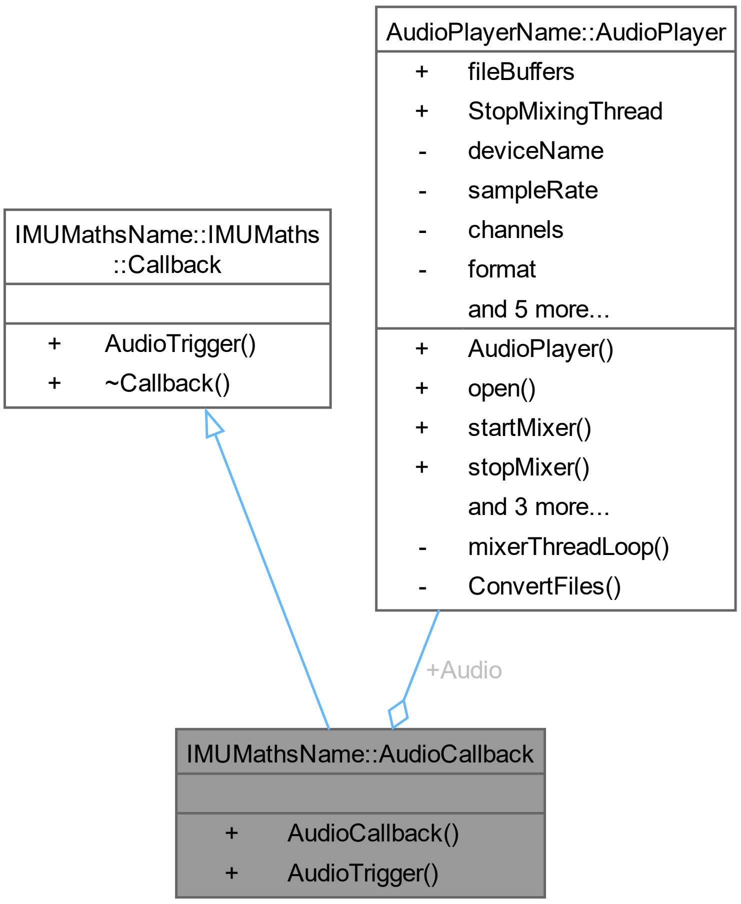
7.2 IMUMathsName::AudioCallback Struct Reference

```
#include <IMUMaths.hpp>
```

Inheritance diagram for IMUMathsName::AudioCallback:



Collaboration diagram for IMUMathsName::AudioCallback:



Public Member Functions

- AudioCallback (AudioPlayerName::AudioPlayer &audio)
- virtual void AudioTrigger (const std::string &FilePath)

Public Member Functions inherited from IMUMathsName::IMUMaths::Callback

- virtual ~Callback ()

Public Attributes

- AudioPlayerName::AudioPlayer & Audio

7.2.1 Constructor & Destructor Documentation

7.2.1.1 AudioCallback()

```
IMUMathsName::AudioCallback::AudioCallback (
            AudioPlayerName::AudioPlayer & audio)  [inline]
```

7.2.2 Member Function Documentation

7.2.2.1 AudioTrigger()

```
virtual void IMUMathsName::AudioCallback::AudioTrigger (
            const std::string & FilePath)  [inline], [virtual]
```

Implements IMUMathsName::IMUMaths::Callback.

7.2.3 Member Data Documentation

7.2.3.1 Audio

```
AudioPlayerName::AudioPlayer& IMUMathsName::AudioCallback::Audio
```

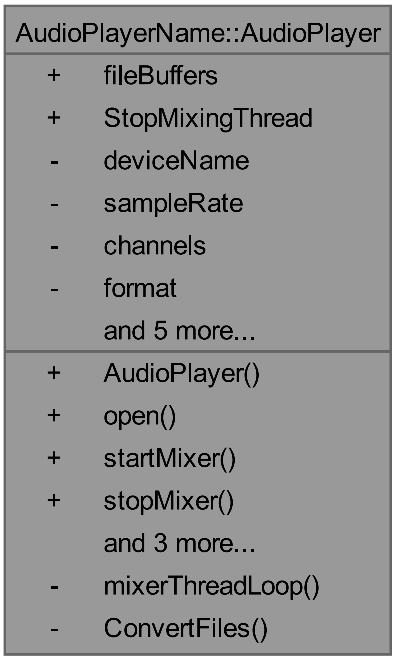
The documentation for this struct was generated from the following file:

- src/libs/IMUMaths/include/IMUMaths.hpp

7.3 AudioPlayerName::AudioPlayer Class Reference

```
#include <ALSAPlayer.hpp>
```

Collaboration diagram for AudioPlayerName::AudioPlayer:



Classes

- struct ActiveSound

Public Member Functions

- AudioPlayer (const std::string &device="default", unsigned int rate=44100, unsigned int ch=2, snd_pcm_↩ format_t fmt=SND_PCM_FORMAT_S16_LE, snd_pcm_uframes_t frames=256, const std::vector< std::string > &filesToConvert={"src/libs/ALSAPlayer/include/CrashCymbal.wav", "src/libs/ALSAPlayer/include/High↩ Tom.wav", "src/libs/ALSAPlayer/include/SnareDrum.wav"})

    Constructor for AudioPlayer class.
- bool open ()

    Open PCM device for playback.
- bool startMixer ()

    Start mixer thread.
- void stopMixer ()

    Stop mixer thread, closes thread when called.
- bool addSoundToMixer (const std::string &fileKey)

    Add input sound to mixer and play it.
- void close ()

    Close PCM handle and free all associated resources.
- ∼AudioPlayer ()

    Destructor.

Public Attributes

- std::unordered_map< std::string, std::vector< int32_t > > fileBuffers
- bool StopMixingThread = false

Private Member Functions

- void mixerThreadLoop ()
- void ConvertFiles (const std::vector< std::string > &filePaths)

Private Attributes

- std::string deviceName
- unsigned int sampleRate
- unsigned int channels
- snd_pcm_format_t format
- snd_pcm_uframes_t framesPerPeriod
- snd_pcm_t * handle
- std::thread mixThread
- std::vector< ActiveSound > ActiveSounds
- std::mutex ActiveMutex

7.3.1 Constructor & Destructor Documentation

7.3.1.1 AudioPlayer()

```
AudioPlayerName::AudioPlayer::AudioPlayer (
             const std::string & device = "default",
             unsigned int rate = 44100,
             unsigned int ch = 2,
             snd_pcm_format_t fmt = SND_PCM_FORMAT_S16_LE,
             snd_pcm_uframes_t frames = 256,
             const std::vector< std::string > & filesToConvert = {"src/libs/ALSAPlayer/include/CrashCymbal.wa
[inline]
```

Constructor for AudioPlayer class.

Handles audio file loading, conversion and playback.

Parameters

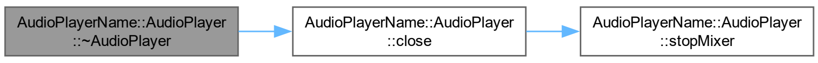
device	The name of the ALSA device to use.
rate	Sample rate in Hz.
ch	Number of channels.
fmt	Format of audio data.
frames	Number of frames per period.
filesToConvert	Sound files used.

7.3.1.2 ∼AudioPlayer()

```
AudioPlayerName::AudioPlayer::∼AudioPlayer ()  [inline]
```

Destructor.

Here is the call graph for this function:


AudioPlayerName::AudioPlayer ::~AudioPlayer
AudioPlayerName::AudioPlayer ::close
AudioPlayerName::AudioPlayer ::stopMixer

7.3.2 Member Function Documentation

7.3.2.1 addSoundToMixer()

```
bool AudioPlayerName::AudioPlayer::addSoundToMixer (
             const std::string & fileKey)  [inline]
```

Add input sound to mixer and play it.

It includes the following steps:

- register detected sound in the mixer
- add sound to buffer and remove sounds that have finished playing
- play sound

Parameters

fileKey	Sound file key.

Returns

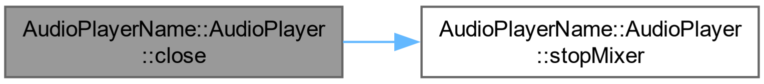
Returns true if sound was added to mixer, false if error

7.3.2.2 close()

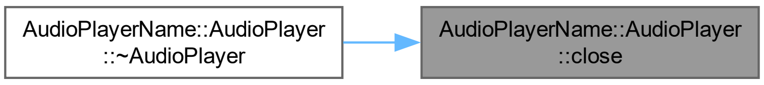
```
void AudioPlayerName::AudioPlayer::close ()  [inline]
```

Close PCM handle and free all associated resources.

Here is the call graph for this function:


AudioPlayerName::AudioPlayer ::close
AudioPlayerName::AudioPlayer ::stopMixer

Here is the caller graph for this function:


AudioPlayerName::AudioPlayer ::~AudioPlayer
AudioPlayerName::AudioPlayer ::close

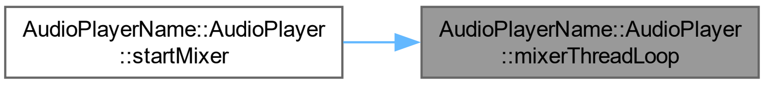
7.3.2.3 ConvertFiles()

```
void AudioPlayerName::AudioPlayer::ConvertFiles (
             const std::vector< std::string > & filePaths)  [inline], [private]
```

7.3.2.4 mixerThreadLoop()

```
void AudioPlayerName::AudioPlayer::mixerThreadLoop ()  [inline], [private]
```

Here is the caller graph for this function:


AudioPlayerName::AudioPlayer ::startMixer
AudioPlayerName::AudioPlayer ::mixerThreadLoop

7.3.2.5 open()

```
bool AudioPlayerName::AudioPlayer::open ()  [inline]
```

Open PCM device for playback.

It includes the following steps:

- open the PCM device
- allocate hardware parameters object and fill it in with default values
- set desired hardware parameters (set access type, format, number of channels, sample rate, period size)
- write parameters to the driver
- get period size

Returns

Returns true is open was successful, but false if something went wrong

7.3.2.6 startMixer()

```
bool AudioPlayerName::AudioPlayer::startMixer ()  [inline]
```

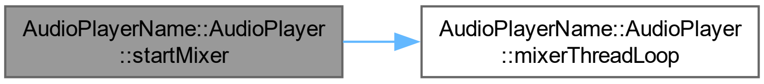
Start mixer thread.

Ensures that a handle exists before opening the mixing thread and tells the user to open the device of not. Sets a boolean to false to control the thread and starts the mixer

Returns

Returns true if mixer thread started correctly, false if error

Here is the call graph for this function:

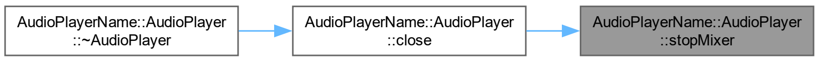

AudioPlayerName::AudioPlayer ::startMixer
AudioPlayerName::AudioPlayer ::mixerThreadLoop

7.3.2.7 stopMixer()

```
void AudioPlayerName::AudioPlayer::stopMixer () [inline]
```

Stop mixer thread, closes thread when called.

Here is the caller graph for this function:


AudioPlayerName::AudioPlayer ::~AudioPlayer
AudioPlayerName::AudioPlayer ::close
AudioPlayerName::AudioPlayer ::stopMixer

7.3.3 Member Data Documentation

7.3.3.1 ActiveMutex

```
std::mutex AudioPlayerName::AudioPlayer::ActiveMutex [private]
```

7.3.3.2 ActiveSounds

```
std::vector<ActiveSound> AudioPlayerName::AudioPlayer::ActiveSounds [private]
```

7.3.3.3 channels

```
unsigned int AudioPlayerName::AudioPlayer::channels [private]
```

7.3.3.4 deviceName

```
std::string AudioPlayerName::AudioPlayer::deviceName [private]
```

7.3.3.5 fileBuffers

```
std::unordered_map<std::string, std::vector<int32_t> > AudioPlayerName::AudioPlayer::file←
Buffers
```

7.3.3.6 format

```
snd_pcm_format_t AudioPlayerName::AudioPlayer::format [private]
```

7.3.3.7 framesPerPeriod

```
snd_pcm_uframes_t AudioPlayerName::AudioPlayer::framesPerPeriod [private]
```

7.3.3.8 handle

```
snd_pcm_t* AudioPlayerName::AudioPlayer::handle  [private]
```

7.3.3.9 mixThread

```
std::thread AudioPlayerName::AudioPlayer::mixThread  [private]
```

7.3.3.10 sampleRate

```
unsigned int AudioPlayerName::AudioPlayer::sampleRate  [private]
```

7.3.3.11 StopMixingThread

```
bool AudioPlayerName::AudioPlayer::StopMixingThread = false
```

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

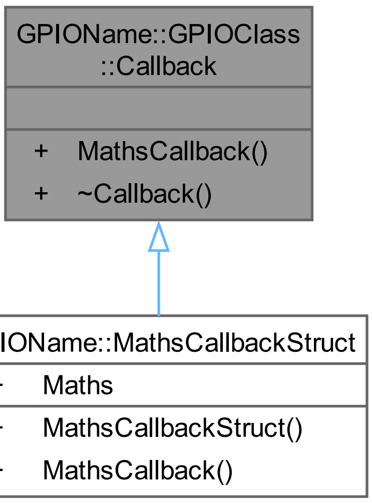
- src/libs/ALSAPlayer/include/ALSAPlayer.hpp

7.4 GPIOName::GPIOClass::Callback Struct Reference

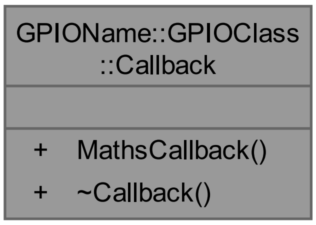
Empty callback to later be filled. Includes destructor.

```
#include <gpioevent.h>
```

Inheritance diagram for GPIOName::GPIOClass::Callback:



Collaboration diagram for GPIOName::GPIOClass::Callback:



Public Member Functions

- virtual void MathsCallback (float X, float Y, float Z)=0
- virtual ∼Callback ()

7.4.1 Detailed Description

Empty callback to later be filled. Includes destructor.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 ∼Callback()

```
virtual GPIOName::GPIOClass::Callback::∼Callback ()  [inline], [virtual]
```

7.4.3 Member Function Documentation

7.4.3.1 MathsCallback()

```
virtual void GPIOName::GPIOClass::Callback::MathsCallback (
             float X,
             float Y,
             float Z)  [pure virtual]
```

Implemented in GPIOName::MathsCallbackStruct.

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

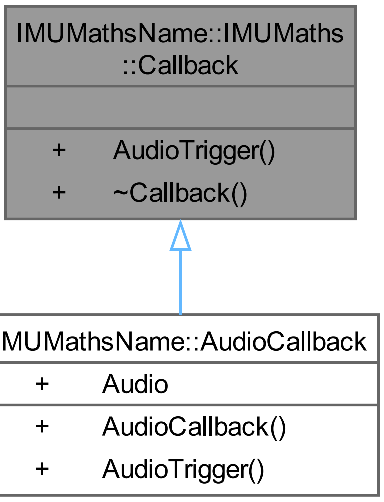
- src/libs/GPIO/include/gpioevent.h

7.5 IMUMathsName::IMUMaths::Callback Struct Reference

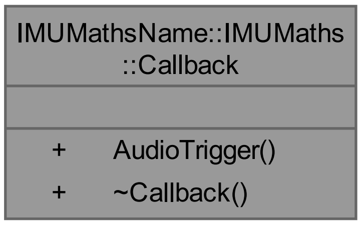
Empty callback to later be filled. Includes destructor.

```
#include <IMUMaths.hpp>
```

Inheritance diagram for IMUMathsName::IMUMaths::Callback:



Collaboration diagram for IMUMathsName::IMUMaths::Callback:



Public Member Functions

- virtual void AudioTrigger (const std::string &FilePath)=0
- virtual ∼Callback ()

7.5.1 Detailed Description

Empty callback to later be filled. Includes destructor.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 ∼Callback()

```
virtual IMUMathsName::IMUMaths::Callback::∼Callback ()  [inline], [virtual]
```

7.5.3 Member Function Documentation

7.5.3.1 AudioTrigger()

```
virtual void IMUMathsName::IMUMaths::Callback::AudioTrigger (
            const std::string & FilePath)  [pure virtual]
```

Implemented in IMUMathsName::AudioCallback.

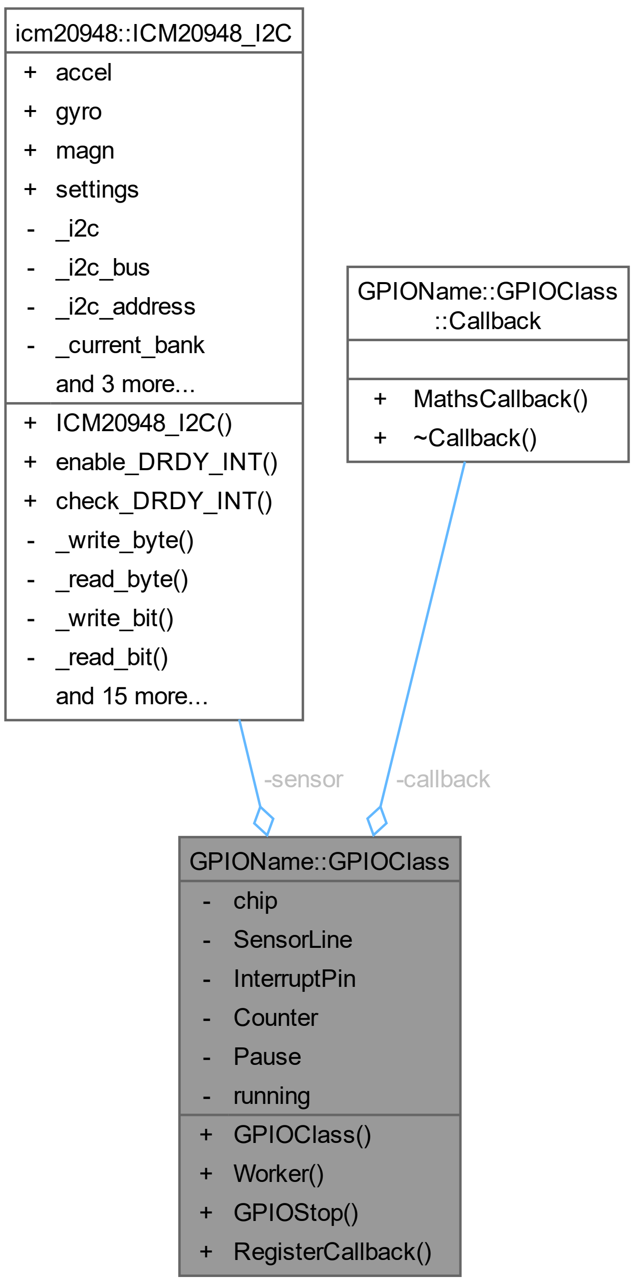
The documentation for this struct was generated from the following file:

- src/libs/IMUMaths/include/IMUMaths.hpp

7.6 GPIOName::GPIOClass Class Reference

```
#include <gpioevent.h>
```

Collaboration diagram for GPIOName::GPIOClass:



Classes

- struct Callback

    Empty callback to later be filled. Includes destructor.

Public Member Functions

- GPIOClass (const char ∗chipName, int InterruptPin, icm20948::ICM20948_I2C &sensor)

Constructor for GPIOClass.

- void Worker ()

  Event driven worker reading data when HIGH seen on GPIO.
- void GPIOStop ()

  Changes a boolean to end the worker.
- void RegisterCallback (Callback ∗cb)

  Registers a callback.

Private Attributes

- gpiod_chip ∗ chip
- gpiod_line ∗ SensorLine
- int InterruptPin
- int Counter
- bool Pause = true
- icm20948::ICM20948_I2C & sensor
- std::atomic< bool > running {true}
- Callback ∗ callback = nullptr

7.6.1 Constructor & Destructor Documentation

7.6.1.1 GPIOClass()

```
GPIOName::GPIOClass::GPIOClass (
            const char * chipName,
            int InterruptPin,
            icm20948::ICM20948_I2C & sensor)
```

Constructor for GPIOClass.

Parameters

chipName	The name of the GPIO chip (e.g., "gpiochip0")
InterruptPin	The GPIO pin number for interrupts
sensor	access to ICM20948_I2C objects

See also

icm20948::ICM20948_I2C

7.6.2 Member Function Documentation

7.6.2.1 GPIOStop()

```
void GPIOName::GPIOClass::GPIOStop ()
```

Changes a boolean to end the worker.

7.6.2.2 RegisterCallback()

```
void GPIOName::GPIOClass::RegisterCallback (
            Callback * cb)  [inline]
```

Registers a callback.

Overwrites the virtual void function within the class with a function taken in via this function

Parameters

cb	callback to register

7.6.2.3 Worker()

```
void GPIOName::GPIOClass::Worker ()
```

Event driven worker reading data when HIGH seen on GPIO.

This function is an event driven interrupt controlled by a GPIO pin. Once this GPIO pin reads HIGH the function will read the data registers using the ReadAccel() callback from the IMU's driver which is then fed into the IMU Maths object to be analysed.

7.6.3 Member Data Documentation

7.6.3.1 callback

```
Callback* GPIOName::GPIOClass::callback = nullptr  [private]
```

7.6.3.2 chip

```
gpiod_chip* GPIOName::GPIOClass::chip  [private]
```

7.6.3.3 Counter

```
int GPIOName::GPIOClass::Counter  [private]
```

7.6.3.4 InterruptPin

```
int GPIOName::GPIOClass::InterruptPin  [private]
```

7.6.3.5 Pause

```
bool GPIOName::GPIOClass::Pause = true  [private]
```

7.6.3.6 running

```
std::atomic<bool> GPIOName::GPIOClass::running {true}  [private]
```

7.6.3.7 sensor

```
icm20948::ICM20948_I2C& GPIOName::GPIOClass::sensor  [private]
```

7.6.3.8 SensorLine

```
gpiod_line* GPIOName::GPIOClass::SensorLine  [private]
```

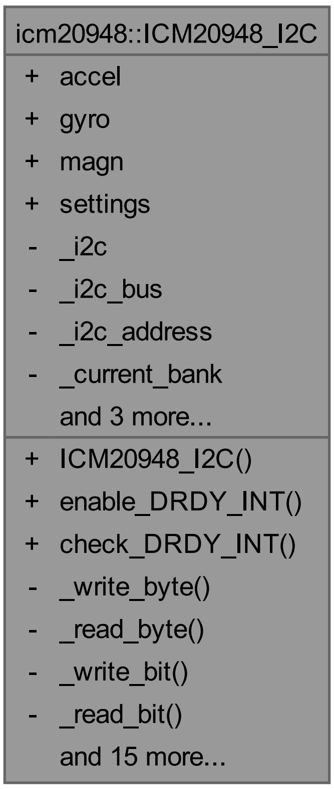
The documentation for this class was generated from the following file:

- src/libs/GPIO/include/gpioevent.h

7.7 icm20948::ICM20948_I2C Class Reference

```
#include <icm20948_i2c.hpp>
```

Collaboration diagram for icm20948::ICM20948_I2C:



Public Member Functions

- ICM20948_I2C (unsigned i2c_bus, unsigned i2c_address=ICM20948_I2C_ADDR, icm20948::settings=icm20948↩ ::settings())

    Constructor for ICM20948_I2C class.
- bool enable_DRDY_INT ()

    Enables the Data Ready Interrupt.
- bool check_DRDY_INT ()

    Checks if the Data Ready Interrupt is active.

Public Attributes

- float accel [3]
- float gyro [3]
- float magn [3]
- icm20948::settings settings

Private Member Functions

- bool _write_byte (const uint8_t bank, const uint8_t reg, const uint8_t byte)
- bool _read_byte (const uint8_t bank, const uint8_t reg, uint8_t &byte)
- bool _write_bit (const uint8_t bank, const uint8_t reg, const uint8_t bit_pos, const bool bit)
- bool _read_bit (const uint8_t bank, const uint8_t reg, const uint8_t bit_pos, bool &bit)
- bool _read_block_bytes (const uint8_t bank, const uint8_t start_reg, uint8_t ∗bytes, const int length)
- bool _write_mag_byte (const uint8_t mag_reg, const uint8_t byte)
- bool _read_mag_byte (const uint8_t mag_reg, uint8_t &byte)
- bool _read_int_byte (const uint8_t bank, const uint8_t reg, uint8_t &byte)
- bool _set_bank (uint8_t bank)
- bool _set_accel_sample_rate_div ()
- bool _set_accel_range_dlpf ()
- bool _set_gyro_sample_rate_div ()
- bool _set_gyro_range_dlpf ()
- bool _magnetometer_init ()
- bool _magnetometer_enable ()
- bool _magnetometer_set_mode ()
- bool _magnetometer_configured ()
- bool _magnetometer_set_readout ()
- bool _chip_i2c_master_reset ()

Private Attributes

- mraa::I2c _i2c
- unsigned _i2c_bus
- unsigned _i2c_address
- uint8_t _current_bank
- float _accel_scale_factor
- float _gyro_scale_factor
- float _magn_scale_factor

7.7.1 Constructor & Destructor Documentation

7.7.1.1 ICM20948_I2C()

```
icm20948::ICM20948_I2C::ICM20948_I2C (
            unsigned i2c_bus,
            unsigned i2c_address = ICM20948_I2C_ADDR,
            icm20948::settings  = icm20948::settings())
```

Constructor for ICM20948_I2C class.

Parameters

i2c_bus	The I2C bus number to which the sensor is connected.
i2c_address	The I2C address of the sensor (default is ICM20948_I2C_ADDR).
settings	The settings structure containing configuration parameters for the sensor. If not provided, default settings will be used

See also

icm20948::settings (external).

7.7.2 Member Function Documentation

7.7.2.1 _chip_i2c_master_reset()

```
bool icm20948::ICM20948_I2C::_chip_i2c_master_reset ()  [private]
```

7.7.2.2 _magnetometer_configured()

```
bool icm20948::ICM20948_I2C::_magnetometer_configured ()  [private]
```

7.7.2.3 _magnetometer_enable()

```
bool icm20948::ICM20948_I2C::_magnetometer_enable ()  [private]
```

7.7.2.4 _magnetometer_init()

```
bool icm20948::ICM20948_I2C::_magnetometer_init ()  [private]
```

7.7.2.5 _magnetometer_set_mode()

```
bool icm20948::ICM20948_I2C::_magnetometer_set_mode ()  [private]
```

7.7.2.6 _magnetometer_set_readout()

```
bool icm20948::ICM20948_I2C::_magnetometer_set_readout ()  [private]
```

7.7.2.7 _read_bit()

```
bool icm20948::ICM20948_I2C::_read_bit (
             const uint8_t bank,
             const uint8_t reg,
             const uint8_t bit_pos,
             bool & bit)  [private]
```

7.7.2.8 _read_block_bytes()

```
bool icm20948::ICM20948_I2C::_read_block_bytes (
            const uint8_t bank,
            const uint8_t start_reg,
            uint8_t * bytes,
            const int length) [private]
```

7.7.2.9 _read_byte()

```
bool icm20948::ICM20948_I2C::_read_byte (
            const uint8_t bank,
            const uint8_t reg,
            uint8_t & byte) [private]
```

7.7.2.10 _read_int_byte()

```
bool icm20948::ICM20948_I2C::_read_int_byte (
            const uint8_t bank,
            const uint8_t reg,
            uint8_t & byte) [private]
```

7.7.2.11 _read_mag_byte()

```
bool icm20948::ICM20948_I2C::_read_mag_byte (
            const uint8_t mag_reg,
            uint8_t & byte) [private]
```

7.7.2.12 _set_accel_range_dlpf()

```
bool icm20948::ICM20948_I2C::_set_accel_range_dlpf () [private]
```

7.7.2.13 _set_accel_sample_rate_div()

```
bool icm20948::ICM20948_I2C::_set_accel_sample_rate_div () [private]
```

7.7.2.14 _set_bank()

```
bool icm20948::ICM20948_I2C::_set_bank (
            uint8_t bank) [private]
```

7.7.2.15 _set_gyro_range_dlpf()

```
bool icm20948::ICM20948_I2C::_set_gyro_range_dlpf () [private]
```

7.7.2.16 _set_gyro_sample_rate_div()

```
bool icm20948::ICM20948_I2C::_set_gyro_sample_rate_div () [private]
```

7.7.2.17 _write_bit()

```
bool icm20948::ICM20948_I2C::_write_bit (
             const uint8_t bank,
             const uint8_t reg,
             const uint8_t bit_pos,
             const bool bit) [private]
```

7.7.2.18 _write_byte()

```
bool icm20948::ICM20948_I2C::_write_byte (
             const uint8_t bank,
             const uint8_t reg,
             const uint8_t byte) [private]
```

7.7.2.19 _write_mag_byte()

```
bool icm20948::ICM20948_I2C::_write_mag_byte (
             const uint8_t mag_reg,
             const uint8_t byte) [private]
```

7.7.2.20 check_DRDY_INT()

```
bool icm20948::ICM20948_I2C::check_DRDY_INT ()
```

Checks if the Data Ready Interrupt is active.

The function is run when the GPIO pin connected to the INT wire recieves a HIGH signal This reads the int_status register, reads the data from the data registers and thus unlatches the interrupt, ready for the next set of data

Returns

true if the registers were successfully read, false if an error occured

7.7.2.21 enable_DRDY_INT()

```
bool icm20948::ICM20948_I2C::enable_DRDY_INT ()
```

Enables the Data Ready Interrupt.

This function enables the Raw Data Ready Interrupt within the IMU by setting the specific registers so that it is notified when new data is available. When new data is available the INT pin on the IMU sends a HIGH value which can be read via a GPIO pin on the Pi.

Returns

true if the setup was successful, false if registers could not be written successefully

7.7.3 Member Data Documentation

7.7.3.1 _accel_scale_factor

```
float icm20948::ICM20948_I2C::_accel_scale_factor  [private]
```

7.7.3.2 _current_bank

```
uint8_t icm20948::ICM20948_I2C::_current_bank  [private]
```

7.7.3.3 _gyro_scale_factor

```
float icm20948::ICM20948_I2C::_gyro_scale_factor  [private]
```

7.7.3.4 _i2c

```
mraa::I2c icm20948::ICM20948_I2C::_i2c  [private]
```

7.7.3.5 _i2c_address

```
unsigned icm20948::ICM20948_I2C::_i2c_address  [private]
```

7.7.3.6 _i2c_bus

```
unsigned icm20948::ICM20948_I2C::_i2c_bus  [private]
```

7.7.3.7 _magn_scale_factor

```
float icm20948::ICM20948_I2C::_magn_scale_factor  [private]
```

7.7.3.8 accel

```
float icm20948::ICM20948_I2C::accel[3]
```

7.7.3.9 gyro

```
float icm20948::ICM20948_I2C::gyro[3]
```

7.7.3.10 magn

```
float icm20948::ICM20948_I2C::magn[3]
```

7.7.3.11 settings

`icm20948::settings icm20948::ICM20948_I2C::settings`

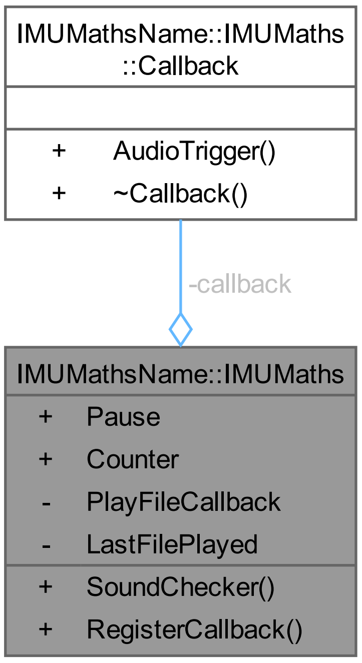
The documentation for this class was generated from the following file:

- src/libs/I2C/include/icm20948_i2c.hpp

7.8 IMUMathsName::IMUMaths Class Reference

`#include <IMUMaths.hpp>`

Collaboration diagram for IMUMathsName::IMUMaths:



Classes

- struct Callback

  Empty callback to later be filled. Includes destructor.

Public Member Functions

- void SoundChecker (float X, float Y, float Z)

  It measures each axis and sees if it falls within desired thresholds.
- void RegisterCallback (Callback ∗cb)

  Registers a callback.

Public Attributes

- bool Pause = false
- int Counter = 0

Private Attributes

- Callback ∗ callback = nullptr
- std::function< void(const std::string &)> PlayFileCallback
- int LastFilePlayed

7.8.1 Member Function Documentation

7.8.1.1 RegisterCallback()

```
void IMUMathsName::IMUMaths::RegisterCallback (
             Callback * cb)  [inline]
```

Registers a callback.

Parameters

cb	callback to register

7.8.1.2 SoundChecker()

```
void IMUMathsName::IMUMaths::SoundChecker (
             float X,
             float Y,
             float Z)
```

It measures each axis and sees if it falls within desired thresholds.

If the acceleration along the specified axis falls within specified thersholds, it will play audio

Parameters

X	acceleration along the x-axis
Y	acceleration along the Y-axis
Z	acceleration along the Z-axis

7.8.2 Member Data Documentation

7.8.2.1 callback

```
Callback* IMUMathsName::IMUMaths::callback = nullptr  [private]
```

7.8.2.2 Counter

```
int IMUMathsName::IMUMaths::Counter = 0
```

7.8.2.3 LastFilePlayed

```
int IMUMathsName::IMUMaths::LastFilePlayed  [private]
```

7.8.2.4 Pause

```
bool IMUMathsName::IMUMaths::Pause = false
```

7.8.2.5 PlayFileCallback

```
std::function<void(const std::string&)> IMUMathsName::IMUMaths::PlayFileCallback  [private]
```

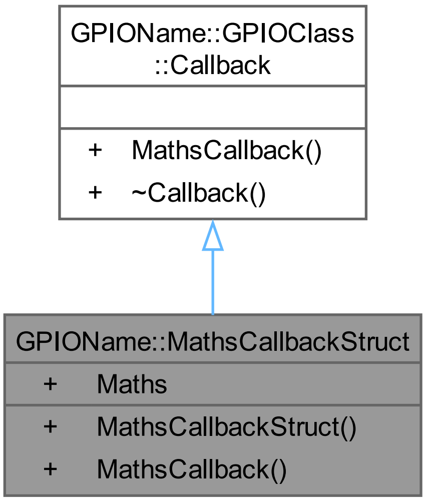
The documentation for this class was generated from the following file:

- src/libs/IMUMaths/include/IMUMaths.hpp

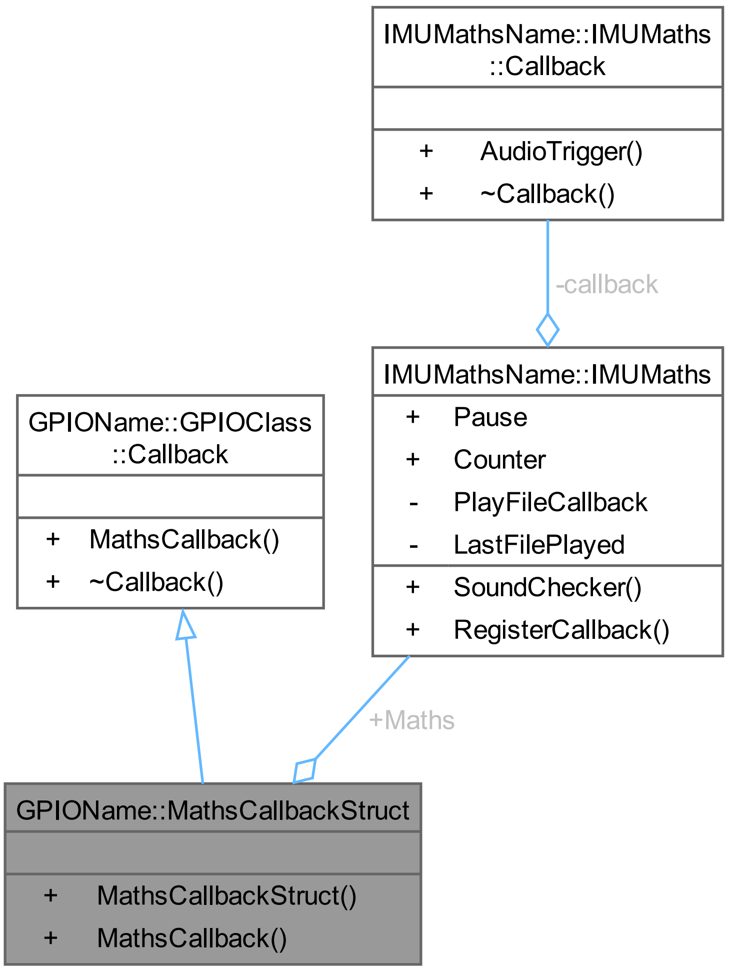
7.9 GPIOName::MathsCallbackStruct Struct Reference

```
#include <gpioevent.h>
```

Inheritance diagram for GPIOName::MathsCallbackStruct:



Collaboration diagram for GPIOName::MathsCallbackStruct:



Public Member Functions

- MathsCallbackStruct (IMUMathsName::IMUMaths &maths)
- virtual void MathsCallback (float X, float Y, float Z) override

Public Member Functions inherited from GPIOName::GPIOClass::Callback

- virtual ∼Callback ()

Public Attributes

- IMUMathsName::IMUMaths & Maths

7.9.1 Constructor & Destructor Documentation

7.9.1.1 MathsCallbackStruct()

```
GPIOName::MathsCallbackStruct::MathsCallbackStruct (
            IMUMathsName::IMUMaths & maths)  [inline]
```

7.9.2 Member Function Documentation

7.9.2.1 MathsCallback()

```
virtual void GPIOName::MathsCallbackStruct::MathsCallback (
            float X,
            float Y,
            float Z)  [inline], [override], [virtual]
```

Implements GPIOName::GPIOClass::Callback.

7.9.3 Member Data Documentation

7.9.3.1 Maths

```
IMUMathsName::IMUMaths& GPIOName::MathsCallbackStruct::Maths
```

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

- src/libs/GPIO/include/gpioevent.h

Chapter 8

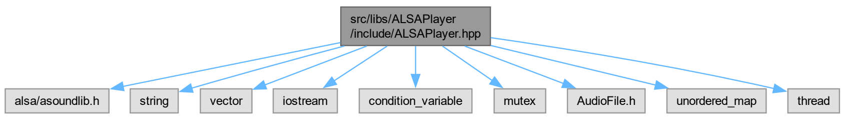
File Documentation

8.1 README.md File Reference

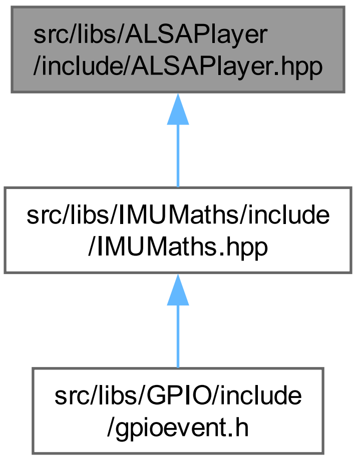
8.2 src/libs/ALSAPlayer/include/ALSAPlayer.hpp File Reference

```
#include <alsa/asoundlib.h>
#include <string>
#include <vector>
#include <iostream>
#include <condition_variable>
#include <mutex>
#include "AudioFile.h"
#include <unordered_map>
#include <thread>
```

Include dependency graph for ALSAPlayer.hpp:



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


src/libs/ALSAPlayer /include/ALSAPlayer.hpp
src/libs/IMUMaths/include /IMUMaths.hpp
src/libs/GPIO/include /gpioevent.h

Classes

- class AudioPlayerName::AudioPlayer
- struct AudioPlayerName::AudioPlayer::ActiveSound

Namespaces

- namespace AudioPlayerName

8.3 ALSAPlayer.hpp

Go to the documentation of this file.

```

#ifndef ALSAPLAYER_H
#define ALSAPLAYER_H

#include <alsa/asoundlib.h>
#include <string>
#include <vector>
#include <iostream>
#include <condition_variable>
#include <mutex>
#include "AudioFile.h"
#include <unordered_map>
#include <thread>


namespace AudioPlayerName{
    class AudioPlayer{
        public:
        std::unordered_map<std::string, std::vector<int32_t>» fileBuffers;

        bool StopMixingThread = false;

        AudioPlayer(const std::string& device="default",
            unsigned int rate = 44100,
            unsigned int ch = 2,
            snd_pcm_format_t fmt = SND_PCM_FORMAT_S16_LE,
            snd_pcm_uframes_t frames = 256,
            const std::vector<std::string>& filesToConvert =
{"src/libs/ALSAPlayer/include/CrashCymbal.wav",
"src/libs/ALSAPlayer/include/HighTom.wav",
"src/libs/ALSAPlayer/include/SnareDrum.wav"})
        : deviceName(device), sampleRate(rate), channels(ch),
        format(fmt), framesPerPeriod(frames), handle(nullptr)
        {
            if (!filesToConvert.empty()){
                ConvertFiles(filesToConvert);
            }
        }

        bool open(){
            int rc = snd_pcm_open(&handle, deviceName.c_str(), SND_PCM_STREAM_PLAYBACK,0);
            if (rc < 0){
                std::cerr « "Unable to open PCM devices: " « snd_strerror(rc) « std::endl;
                return false;
            }

            snd_pcm_hw_params_t* params;
            snd_pcm_hw_params_alloca(&params);
            snd_pcm_hw_params_any(handle, params);
            snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
            snd_pcm_hw_params_set_format(handle, params, format);
            snd_pcm_hw_params_set_channels(handle, params, channels);

            unsigned int rate_near = sampleRate;
            snd_pcm_hw_params_set_rate_near(handle, params, &rate_near,0);

            rc = snd_pcm_hw_params_set_period_size_near(handle, params, &framesPerPeriod, 0);
            if (rc <0) {
                std::cerr « "Unable to set HW parameters: " « snd_strerror(rc) « std::endl;
                return false;
            }
            snd_pcm_uframes_t bufferSize = framesPerPeriod * 4;
            rc = snd_pcm_hw_params_set_buffer_size_near(handle, params, &bufferSize);
            if (rc < 0) {
                std::cerr « "Unable to set buffer size: " « snd_strerror(rc) « std::endl;
                return false;
            }

            rc = snd_pcm_hw_params(handle, params);
            if (rc < 0) {
             std::cerr « "Unable to set HW parameters: " « snd_strerror(rc) « std::endl;
             return false;
            }


            // Verify the final chosen period size and buffer size
            snd_pcm_hw_params_get_period_size(params, &framesPerPeriod, 0);
            snd_pcm_hw_params_get_buffer_size(params, &bufferSize);
            std::cout « "[DEBUG] Final period size: " « framesPerPeriod « std::endl;
            std::cout « "[DEBUG] Final buffer size: " « bufferSize « std::endl;

            return true;
        }


        bool startMixer() {
            if (!handle) {
                std::cerr « "ALSA device is not open. Call open() first." « std::endl;
                return false;
            }
            StopMixingThread = false;
            mixThread = std::thread(&AudioPlayer::mixerThreadLoop, this);

            return true;
        }

        void stopMixer() {
            StopMixingThread = true;
            if (mixThread.joinable()) {
                mixThread.join();
            }
        }

        bool addSoundToMixer(const std::string& fileKey) {
            std::lock_guard<std::mutex> lock(ActiveMutex);

            // Check if file buffer exists
            auto it = fileBuffers.find(fileKey);
            if (it == fileBuffers.end()) {
```

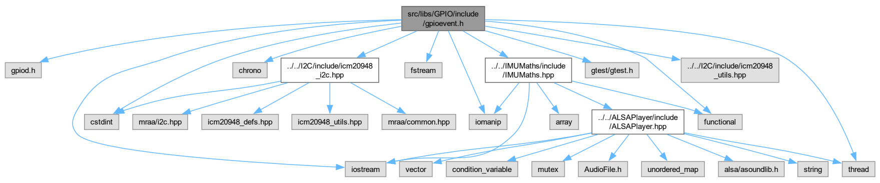
```
            std::cerr « "Audio buffer not found for file: " « fileKey « std::endl;
            return false;
        }

        // Create a new ActiveSound
        ActiveSound newSound;

        // pointer to the file’s buffer
        newSound.buffer = &it->second;
        newSound.position = 0;

        // Add it to active sounds
        ActiveSounds.push_back(newSound);
        return true;
    }

    void close() {
        stopMixer();
        if (handle) {
            snd_pcm_drop(handle);
            snd_pcm_close(handle);
            handle = nullptr;
        }
    }

    ~AudioPlayer() {
        close();
    }


    private:
    std::string deviceName;
    unsigned int sampleRate;
    unsigned int channels;
    snd_pcm_format_t format;
    snd_pcm_uframes_t framesPerPeriod;
    snd_pcm_t* handle;

    std::thread mixThread;

    struct ActiveSound {
        std::vector<int32_t>* buffer;
        size_t position;
    };

    std::vector<ActiveSound> ActiveSounds;
    std::mutex ActiveMutex;

    void mixerThreadLoop() {
        // Allocate a buffer for one period of audio
        const size_t periodSizeSamples = framesPerPeriod * channels;
        std::vector<int32_t> mixBuffer(periodSizeSamples, 0);

        while (!StopMixingThread) {
            // Clear the mix buffer each iteration
            std::fill(mixBuffer.begin(), mixBuffer.end(), 0);

            {
                // Locks the active list ofsounds
                std::lock_guard<std::mutex> lock(ActiveMutex);

                // Mixes all of the activesounds and removes those that have finished
                for (auto it = ActiveSounds.begin(); it != ActiveSounds.end(); ) {
                    ActiveSound& sound = *it;
                    const size_t totalFrames = sound.buffer->size() / channels;
                    size_t framesLeft = totalFrames - sound.position;

                    // Callculate how may frakes are left to be mixed
                    size_t framesToMix = std::min<size_t>(framesPerPeriod, framesLeft);

                    //Mix the audio data from this sound into the buffer
                    for (size_t f = 0; f < framesToMix; ++f) {
                        for (unsigned int c = 0; c < channels; ++c) {
                            // Source index in the file buffer
                            size_t srcIndex = (sound.position + f) * channels + c;
                            // Destination index in the mix buffer
                            size_t dstIndex = f * channels + c;

                            //Add up the sample
                            mixBuffer[dstIndex] += (*sound.buffer)[srcIndex];
                        }
                    }

                    // Advance playback position
                    sound.position += framesToMix;

                    // If a sound has finished, remove it
```

```
                        if (sound.position >= totalFrames) {
                            it = ActiveSounds.erase(it);
                        } else {
                            ++it;
                        }
                    }
                }

                // Write the mixed buffer to ALSA
                int rc = snd_pcm_writei(handle, mixBuffer.data(), framesPerPeriod);
                if (rc == -EPIPE) {
                    std::cerr « "Underrun occurred\n";
                    snd_pcm_prepare(handle);
                } else if (rc < 0) {
                    std::cerr « "Error from writei: " « snd_strerror(rc) « std::endl;
                }
            }
        }


        void ConvertFiles(const std::vector<std::string>& filePaths) {
            std::vector<int32_t> result;

            for (const auto& path : filePaths) {
                AudioFile<int32_t> file;
                if (!file.load(path)) {
                    std::cerr « "Error loading file: " « path « std::endl;
                    continue;
                }

                int fileChannels = file.getNumChannels();
                int ChannelSamples = file.getNumSamplesPerChannel();

                std::vector<int32_t> interleaved;
                interleaved.reserve(ChannelSamples * fileChannels);
                for (int i=0; i < ChannelSamples; ++i){
                    for (int ch = 0; ch < fileChannels; ++ch){
                        interleaved.push_back(file.samples[ch][i]);

                    }


                }
                fileBuffers[path] = std::move(interleaved);

            }
        }


    };
}



#endif
```

8.4 src/libs/GPIO/include/gpioevent.h File Reference

```
#include <gpiod.h>
#include <iostream>
#include <thread>
#include <chrono>
#include <iomanip>
#include <fstream>
#include <cstdint>
#include <functional>
#include <gtest/gtest.h>
#include "../../I2C/include/icm20948_i2c.hpp"
#include "../../I2C/include/icm20948_utils.hpp"
#include "../../IMUMaths/include/IMUMaths.hpp"
```

Include dependency graph for gpioevent.h:



Classes

- class GPIOName::GPIOClass
- struct GPIOName::GPIOClass::Callback

    Empty callback to later be filled. Includes destructor.
- struct GPIOName::MathsCallbackStruct

Namespaces

- namespace GPIOName

8.5 gpioevent.h

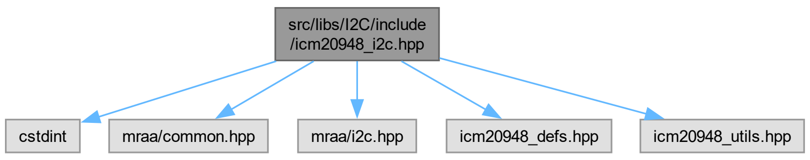
Go to the documentation of this file.

```
#ifndef GPIOEVENT_H
#define GPIOEVENT_H


#include <gpiod.h>
#include <iostream>
#include <thread>
#include <chrono>
#include <iomanip>
#include <fstream>
#include <cstdint>
#include <functional>
#include <gtest/gtest.h>

#include "../../I2C/include/icm20948_i2c.hpp"
#include "../../I2C/include/icm20948_utils.hpp"

#include "../../IMUMaths/include/IMUMaths.hpp"


namespace GPIOName {
    class GPIOClass {
    public:

        GPIOClass(const char* chipName, int InterruptPin,
             icm20948::ICM20948_I2C& sensor);

        void Worker();

        void GPIOStop();

        struct Callback{
            virtual void MathsCallback(float X, float Y, float Z) = 0;
            virtual ~Callback(){};
        };

        void RegisterCallback(Callback* cb){
            callback = cb;
        }

        //Testing private stuff
        #ifdef UNIT_TEST
            bool GetRunning() const {
                return running.load();
            }

            bool HasCallback() const {
                return callback != nullptr;
            }

            Callback* GetCallback() const {
                return callback;
            }

        #endif

        private:
        gpiod_chip* chip;
        gpiod_line* SensorLine;
        int InterruptPin;
        int Counter;
        bool Pause = true;

        //Reference to IMU's driver
        icm20948::ICM20948_I2C& sensor;

        std::atomic<bool> running{true};
        Callback* callback = nullptr;

    };

    struct MathsCallbackStruct : GPIOName::GPIOClass::Callback{
        IMUMathsName::IMUMaths& Maths;

        MathsCallbackStruct(IMUMathsName::IMUMaths& maths) : Maths(maths) {}

        virtual void MathsCallback(float X, float Y, float Z) override {
            Maths.SoundChecker(X, Y, Z);
        }
    };
}



#endif
```

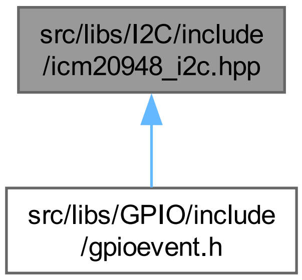
8.6 src/libs/I2C/include/icm20948_i2c.hpp File Reference

```
#include <cstdint>
#include "mraa/common.hpp"
#include "mraa/i2c.hpp"
#include "icm20948_defs.hpp"
#include "icm20948_utils.hpp"
```

Include dependency graph for icm20948_i2c.hpp:



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



Classes

- class icm20948::ICM20948_I2C

Namespaces

- namespace icm20948

8.7 icm20948_i2c.hpp

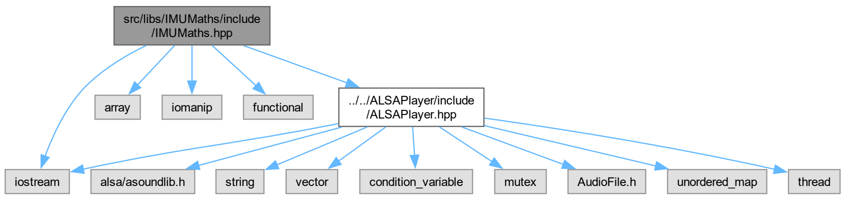
Go to the documentation of this file.

```
#ifndef ICM20948_I2C_HPP
#define ICM20948_I2C_HPP

#include <cstdint>

#include "mraa/common.hpp"
#include "mraa/i2c.hpp"

#include "icm20948_defs.hpp"
#include "icm20948_utils.hpp"

namespace icm20948
{
    class ICM20948_I2C
    {
        private:
            mraa::I2c _i2c;
            unsigned _i2c_bus, _i2c_address;
            uint8_t _current_bank;
            float _accel_scale_factor, _gyro_scale_factor, _magn_scale_factor;

            bool _write_byte(const uint8_t bank, const uint8_t reg, const uint8_t byte);
            bool _read_byte(const uint8_t bank, const uint8_t reg, uint8_t &byte);
            bool _write_bit(const uint8_t bank, const uint8_t reg, const uint8_t bit_pos, const bool
 bit);
            bool _read_bit(const uint8_t bank, const uint8_t reg, const uint8_t bit_pos, bool &bit);
            bool _read_block_bytes(const uint8_t bank, const uint8_t start_reg, uint8_t *bytes, const
 int length);
            bool _write_mag_byte(const uint8_t mag_reg, const uint8_t byte);
            bool _read_mag_byte(const uint8_t mag_reg, uint8_t &byte);
            bool _read_int_byte(const uint8_t bank, const uint8_t reg, uint8_t &byte);

            bool _set_bank(uint8_t bank);
            bool _set_accel_sample_rate_div();
            bool _set_accel_range_dlpf();
            bool _set_gyro_sample_rate_div();
            bool _set_gyro_range_dlpf();

            bool _magnetometer_init();
            bool _magnetometer_enable();
            bool _magnetometer_set_mode();
            bool _magnetometer_configured();
            bool _magnetometer_set_readout();

            bool _chip_i2c_master_reset();

        public:
            // Contains linear acceleration in m/s^2
            float accel[3];
            // Contains angular velocities in rad/s
            float gyro[3];
            // Contains magnetic field strength in uTesla
            float magn[3];

            // Sensor settings
            icm20948::settings settings;

            ICM20948_I2C(unsigned i2c_bus, unsigned i2c_address = ICM20948_I2C_ADDR,
icm20948::settings
                 = icm20948::settings());


            bool init();

            bool reset();

            bool wake();

            bool set_settings();

            bool read_accel_gyro();

            bool read_magn();


            bool enable_DRDY_INT();

            bool check_DRDY_INT();
    };
}

#endif
```

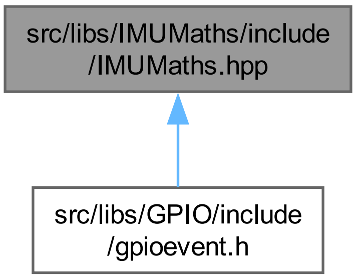
8.8 src/libs/IMUMaths/include/IMUMaths.hpp File Reference

```
#include <iostream>
#include <array>
#include <iomanip>
#include <functional>
#include "../../ALSAPlayer/include/ALSAPlayer.hpp"
```

Include dependency graph for IMUMaths.hpp:



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



Classes

- class IMUMathsName::IMUMaths
- struct IMUMathsName::IMUMaths::Callback

  Empty callback to later be filled. Includes destructor.
- struct IMUMathsName::AudioCallback

Namespaces

- namespace IMUMathsName

8.9 IMUMaths.hpp

Go to the documentation of this file.

```
#ifndef IMUMATHS_H
#define IMUMATHS_H


#include <iostream>
#include <array>
#include <iomanip>
#include <functional>

#include "../../ALSAPlayer/include/ALSAPlayer.hpp"


namespace IMUMathsName {
    class IMUMaths{

        public:

        void SoundChecker(float X, float Y, float Z);

        // Pauses
        bool Pause = false;

        // Counter variable
        int Counter = 0;

        struct Callback{
            virtual void AudioTrigger(const std::string& FilePath) = 0;
            virtual ~Callback(){};
        };


        void RegisterCallback(Callback* cb){
            callback = cb;
            //std::cout « "[IMUMaths] Registered callback at address: " « callback « std::endl;
        }

        //Access to private fo UNIT_TEST only
        #ifdef UNIT_TEST

        bool HasCallback() const {
            return callback != nullptr;
        }

        Callback* GetCallback() const {
            return callback;
        }

        int LastFilePlayedTest() const{
            return LastFilePlayed;
        }

        #endif


        private:

        Callback* callback = nullptr;
        std::function<void(const std::string&)> PlayFileCallback;

        // For debugging: Identifier of the last audio file played

        int LastFilePlayed;


    };

    struct AudioCallback : IMUMathsName::IMUMaths::Callback{
        AudioPlayerName::AudioPlayer& Audio;

        AudioCallback(AudioPlayerName::AudioPlayer& audio) : Audio(audio) {}

        virtual void AudioTrigger(const std::string& FilePath){
            std::thread([this,FilePath]{
                Audio.addSoundToMixer(FilePath);

            }).detach();
        }
    };
}


#endif
```