

# SnAirBeats

1.0

Generated by Doxygen 1.13.2



<b>1 SnAirBeats</b>	<b>1</b>
1.1 SnAIRbeats	1
1.2 Building	1
1.3 Prerequisites	2
1.4 Compilation from source	2
1.5 Usage	2
1.5.1 Maximum Latency	3
1.6 Libraries	3
1.6.1 ALSAPlayer	3
1.6.2 GPIO	3
1.6.3 I2C	3
1.6.4 IMUMaths	4
1.7 Unit tests	4
1.8 Documentation	4
1.9 Sponsorship and funding	4
1.10 Media	4
1.11 Authors	4
1.12 Licenses	4
<b>2 Namespace Index</b>	<b>5</b>
2.1 Namespace List	5
<b>3 Hierarchical Index</b>	<b>7</b>
3.1 Class Hierarchy	7
<b>4 Class Index</b>	<b>9</b>
4.1 Class List	9
<b>5 File Index</b>	<b>11</b>
5.1 File List	11
<b>6 Namespace Documentation</b>	<b>13</b>
6.1 AudioPlayerName Namespace Reference	13
6.2 GPIOName Namespace Reference	13
6.2.1 Typedef Documentation	13
6.2.1.1 GPIOCallback	13
6.3 icm20948 Namespace Reference	13
6.4 IMUMathsName Namespace Reference	13
<b>7 Class Documentation</b>	<b>15</b>
7.1 AudioPlayerName::AudioPlayer::ActiveSound Struct Reference	15
7.1.1 Member Data Documentation	15
7.1.1.1 buffer	15
7.1.1.2 position	15

7.2 IMUMathsName::AudioCallback Struct Reference	16
7.2.1 Constructor & Destructor Documentation	18
7.2.1.1 AudioCallback()	18
7.2.2 Member Function Documentation	18
7.2.2.1 AudioTrigger()	18
7.2.3 Member Data Documentation	18
7.2.3.1 Audio	18
7.3 AudioPlayerName::AudioPlayer Class Reference	18
7.3.1 Constructor & Destructor Documentation	20
7.3.1.1 AudioPlayer()	20
7.3.1.2 ~AudioPlayer()	21
7.3.2 Member Function Documentation	21
7.3.2.1 addSoundToMixer()	21
7.3.2.2 close()	21
7.3.2.3 ConvertFiles()	22
7.3.2.4 mixerThreadLoop()	22
7.3.2.5 open()	22
7.3.2.6 startMixer()	23
7.3.2.7 stopMixer()	23
7.3.3 Member Data Documentation	23
7.3.3.1 ActiveMutex	23
7.3.3.2 ActiveSounds	23
7.3.3.3 channels	23
7.3.3.4 deviceName	24
7.3.3.5 fileBuffers	24
7.3.3.6 format	24
7.3.3.7 framesPerPeriod	24
7.3.3.8 handle	24
7.3.3.9 mixThread	24
7.3.3.10 sampleRate	24
7.3.3.11 StopMixingThread	24
7.4 GPIOName::GPIOClass::Callback Struct Reference	25
7.4.1 Detailed Description	26
7.4.2 Constructor & Destructor Documentation	26
7.4.2.1 ~Callback()	26
7.4.3 Member Function Documentation	26
7.4.3.1 MathsCallback()	26
7.5 IMUMathsName::IMUMaths::Callback Struct Reference	26
7.5.1 Detailed Description	27
7.5.2 Constructor & Destructor Documentation	27
7.5.2.1 ~Callback()	27
7.5.3 Member Function Documentation	27

7.5.3.1 AudioTrigger()	27
7.6 GPIOName::GPIOClass Class Reference	28
7.6.1 Constructor & Destructor Documentation	29
7.6.1.1 GPIOClass()	29
7.6.2 Member Function Documentation	30
7.6.2.1 GPIOStop()	30
7.6.2.2 IsRunning()	30
7.6.2.3 RegisterCallback()	30
7.6.2.4 Worker()	30
7.6.2.5 WorkerDataCollect()	30
7.6.3 Member Data Documentation	31
7.6.3.1 callback	31
7.6.3.2 chip	31
7.6.3.3 Counter	31
7.6.3.4 InterruptPin	31
7.6.3.5 LEDLine	31
7.6.3.6 Maths	31
7.6.3.7 Pause	31
7.6.3.8 running	31
7.6.3.9 sensor	31
7.6.3.10 SensorLine	32
7.7 icm20948::ICM20948_I2C Class Reference	32
7.7.1 Constructor & Destructor Documentation	33
7.7.1.1 ICM20948_I2C()	33
7.7.2 Member Function Documentation	34
7.7.2.1 _chip_i2c_master_reset()	34
7.7.2.2 _magnetometer_configured()	34
7.7.2.3 _magnetometer_enable()	34
7.7.2.4 _magnetometer_init()	34
7.7.2.5 _magnetometer_set_mode()	34
7.7.2.6 _magnetometer_set_readout()	34
7.7.2.7 _read_bit()	34
7.7.2.8 _read_block_bytes()	35
7.7.2.9 _read_byte()	35
7.7.2.10 _read_int_byte()	35
7.7.2.11 _read_mag_byte()	35
7.7.2.12 _set_accel_range_dlpf()	35
7.7.2.13 _set_accel_sample_rate_div()	35
7.7.2.14 _set_bank()	35
7.7.2.15 _set_gyro_range_dlpf()	35
7.7.2.16 _set_gyro_sample_rate_div()	36
7.7.2.17 _write_bit()	36

7.7.2.18 _write_byte()	36
7.7.2.19 _write_mag_byte()	36
7.7.2.20 check_DRDY_INT()	36
7.7.2.21 enable_DRDY_INT()	36
7.7.3 Member Data Documentation	37
7.7.3.1 _accel_scale_factor	37
7.7.3.2 _current_bank	37
7.7.3.3 _gyro_scale_factor	37
7.7.3.4 _i2c	37
7.7.3.5 _i2c_address	37
7.7.3.6 _i2c_bus	37
7.7.3.7 _magn_scale_factor	37
7.7.3.8 accel	37
7.7.3.9 gyro	37
7.7.3.10 magn	37
7.7.3.11 settings	38
7.8 IMUMathsName::IMUMaths Class Reference	38
7.8.1 Constructor & Destructor Documentation	39
7.8.1.1 IMUMaths()	39
7.8.1.2 ~IMUMaths()	40
7.8.2 Member Function Documentation	40
7.8.2.1 RegisterCallback()	40
7.8.2.2 SetPlayFileCallback()	40
7.8.2.3 SoundChecker()	40
7.8.3 Member Data Documentation	41
7.8.3.1 Audio	41
7.8.3.2 callback	41
7.8.3.3 Counter	41
7.8.3.4 LastFilePlayed	41
7.8.3.5 Pause	41
7.8.3.6 PlayFileCallback	41
7.9 GPIOName::MathsCallbackStruct Struct Reference	42
7.9.1 Constructor & Destructor Documentation	44
7.9.1.1 MathsCallbackStruct()	44
7.9.2 Member Function Documentation	44
7.9.2.1 MathsCallback()	44
7.9.3 Member Data Documentation	44
7.9.3.1 Maths	44
<b>8 File Documentation</b>	<b>45</b>
8.1 README.md File Reference	45
8.2 src/libs/ALSAPlayer/include/ALSAPlayer.hpp File Reference	45

---

8.3 ALSAPlayer.hpp . . . . .	46
8.4 src/libs/GPIO/include/gpioevent.h File Reference . . . . .	49
8.5 gpioevent.h . . . . .	50
8.6 src/libs/I2C/include/icm20948_i2c.hpp File Reference . . . . .	51
8.7 icm20948_i2c.hpp . . . . .	52
8.8 src/libs/IMUMaths/include/IMUMaths.hpp File Reference . . . . .	53
8.9 IMUMaths.hpp . . . . .	54





# Chapter 1

## SnAirBeats



### 1.1 SnAIRbeats

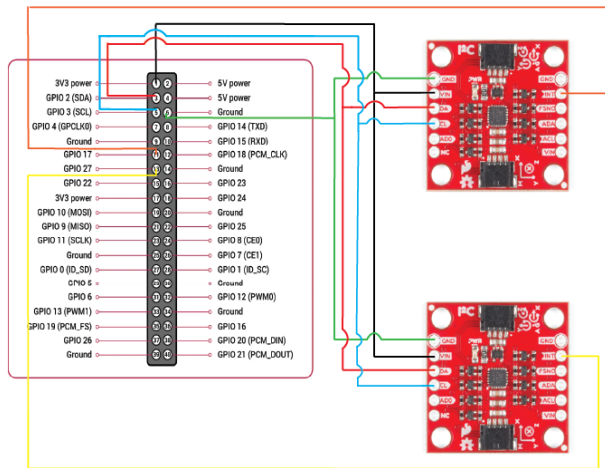
SnAirBeats is a next generation method to practice the drums, while reducing noise and space typically required to do so. The SnAirBeat set uses inertial measurement units (IMU) within the sticks to track their movement and play a corresponding drum, not requiring any physical hitting like modern electric 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 recommended to use a Raspberry Pi operating system such as [Raspbian](#) 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 Compilation 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](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, most 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 library 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](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:

<a href="#">AudioPlayerName</a>	13
<a href="#">GPIOName</a>	13
<a href="#">icm20948</a>	13
<a href="#">IMUMathsName</a>	13





## Chapter 3

# Hierarchical Index

### 3.1 Class Hierarchy

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

AudioPlayerName::AudioPlayer::ActiveSound . . . . .	15
AudioPlayerName::AudioPlayer . . . . .	18
GPIOName::GPIOClass::Callback . . . . .	25
GPIOName::MathsCallbackStruct . . . . .	42
IMUMathsName::IMUMaths::Callback . . . . .	26
IMUMathsName::AudioCallback . . . . .	16
GPIOName::GPIOClass . . . . .	28
icm20948::ICM20948_I2C . . . . .	32
IMUMathsName::IMUMaths . . . . .	38



## Chapter 4

# Class Index

### 4.1 Class List

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

<a href="#">AudioPlayerName::AudioPlayer::ActiveSound</a>	15
<a href="#">IMUMathsName::AudioCallback</a>	16
<a href="#">AudioPlayerName::AudioPlayer</a>	18
<a href="#">GPIOName::GPIOClass::Callback</a>	
<a href="#">Callback</a> using virtual void	25
<a href="#">IMUMathsName::IMUMaths::Callback</a>	
<a href="#">Callback</a> using virtual void	26
<a href="#">GPIOName::GPIOClass</a>	28
<a href="#">icm20948::ICM20948_I2C</a>	32
<a href="#">IMUMathsName::IMUMaths</a>	38
<a href="#">GPIOName::MathsCallbackStruct</a>	42



## Chapter 5

# File Index

### 5.1 File List

Here is a list of all files with brief descriptions:

src/libs/ALSAPlayer/include/ <a href="#">ALSAPlayer.hpp</a> . . . . .	45
src/libs/GPIO/include/ <a href="#">gpioevent.h</a> . . . . .	49
src/libs/I2C/include/ <a href="#">icm20948_i2c.hpp</a> . . . . .	51
src/libs/IMUMaths/include/ <a href="#">IMUMaths.hpp</a> . . . . .	53



## Chapter 6

# Namespace Documentation

### 6.1 AudioPlayerName Namespace Reference

#### Classes

- class [AudioPlayer](#)

### 6.2 GPIOName Namespace Reference

#### Classes

- class [GPIOClass](#)
- struct [MathsCallbackStruct](#)

#### Typedefs

- typedef void(\* [GPIOCallback](#)) (void \*context, float, float, float)

#### 6.2.1 Typedef Documentation

##### 6.2.1.1 GPIOCallback

```
typedef void(* GPIOName::GPIOCallback) (void *context, float, float, float)
```

### 6.3 icm20948 Namespace Reference

#### Classes

- class [ICM20948\\_I2C](#)

### 6.4 IMUMathsName Namespace Reference

#### Classes

- struct [AudioCallback](#)
- class [IMUMaths](#)



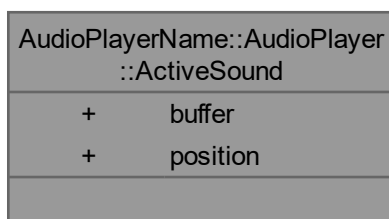


# Chapter 7

## 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
```

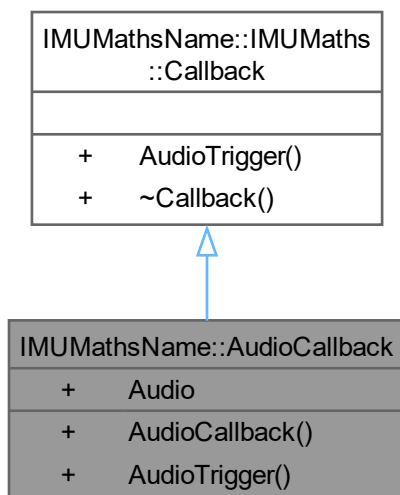
The documentation for this struct was generated from the following file:

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

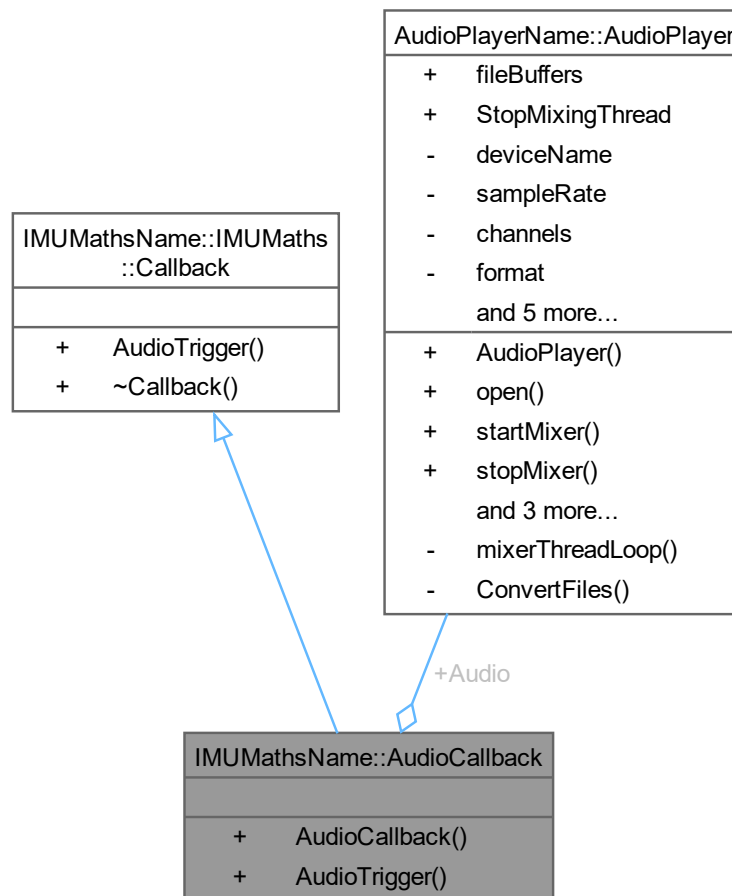
## 7.2 IMUMathsName::AudioCallback Struct Reference

```
#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:

AudioPlayerName::AudioPlayer	
+	fileBuffers
	StopMixingThread
	- deviceName
	- sampleRate
	- channels
	- format
	and 5 more...
+	AudioPlayer()
	open()
	startMixer()
	stopMixer()
	and 3 more...
	- mixerThreadLoop()
	- ConvertFiles()

## 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/HighTom.wav", "src/libs/ALSAPlayer/include/SnareDrum.wav"})  
*Constructor for [AudioPlayer](#) class.*
- bool [open](#) ()  
*Open PCM device for playback.*
- void [startMixer](#) ()  
*Start mixer thread.*
- void [stopMixer](#) ()  
*Stop mixer thread.*
- 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

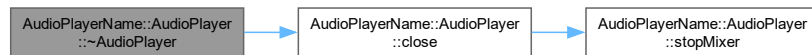
<i>device</i>	The name of the ALSA device to use.
<i>rate</i>	Sample rate in Hz.
<i>ch</i>	Number of channels.
<i>fmt</i>	Format of audio data.
<i>frames</i>	Number of frames per period.
<i>filesToConvert</i>	Sound files used.

### 7.3.1.2 ~AudioPlayer()

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

Destructor.

Here is the call graph for this function:



## 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

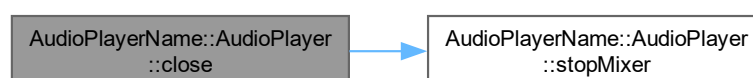
<i>fileKey</i>	Sound file key.
----------------	-----------------

### 7.3.2.2 close()

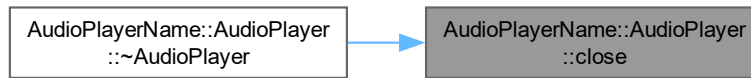
```
void AudioPlayerName::AudioPlayer::close () [inline]
```

Close PCM handle and free all associated resources.

Here is the call graph for this function:



Here is the caller graph for this function:



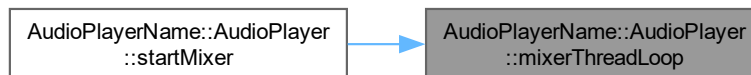
### 7.3.2.3 ConvertFiles()

```
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:



### 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



### 7.3.2.6 startMixer()

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

Start mixer thread.

Here is the call graph for this function:

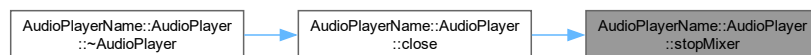


### 7.3.2.7 stopMixer()

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

Stop mixer thread.

Here is the caller graph for this function:



## 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:

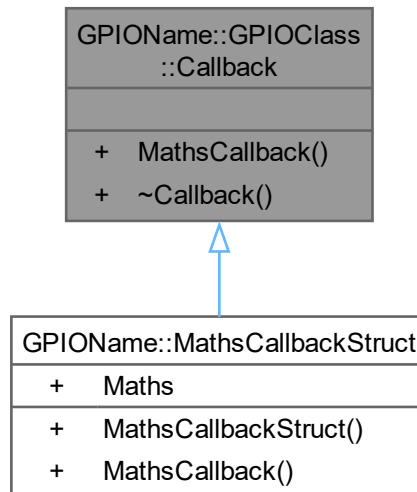
- src/libs/ALSAPlayer/include/[ALSAPlayer.hpp](#)

## 7.4 GPIOName::GPIOClass::Callback Struct Reference

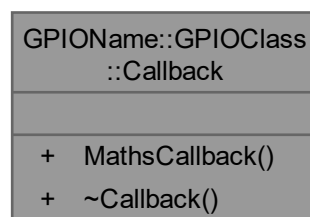
[Callback](#) using virtual void.

```
#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

[Callback](#) using virtual void.

### 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:

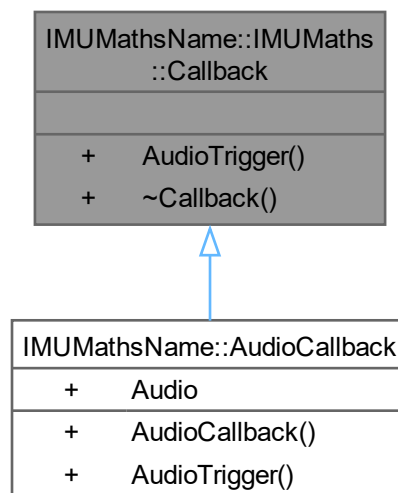
- [src/libs/GPIO/include/gpioevent.h](#)

## 7.5 IMUMathsName::IMUMaths::Callback Struct Reference

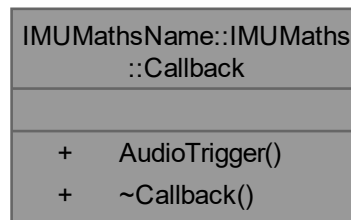
[Callback](#) using virtual void.

```
#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

[Callback](#) using virtual void.

### 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](#).

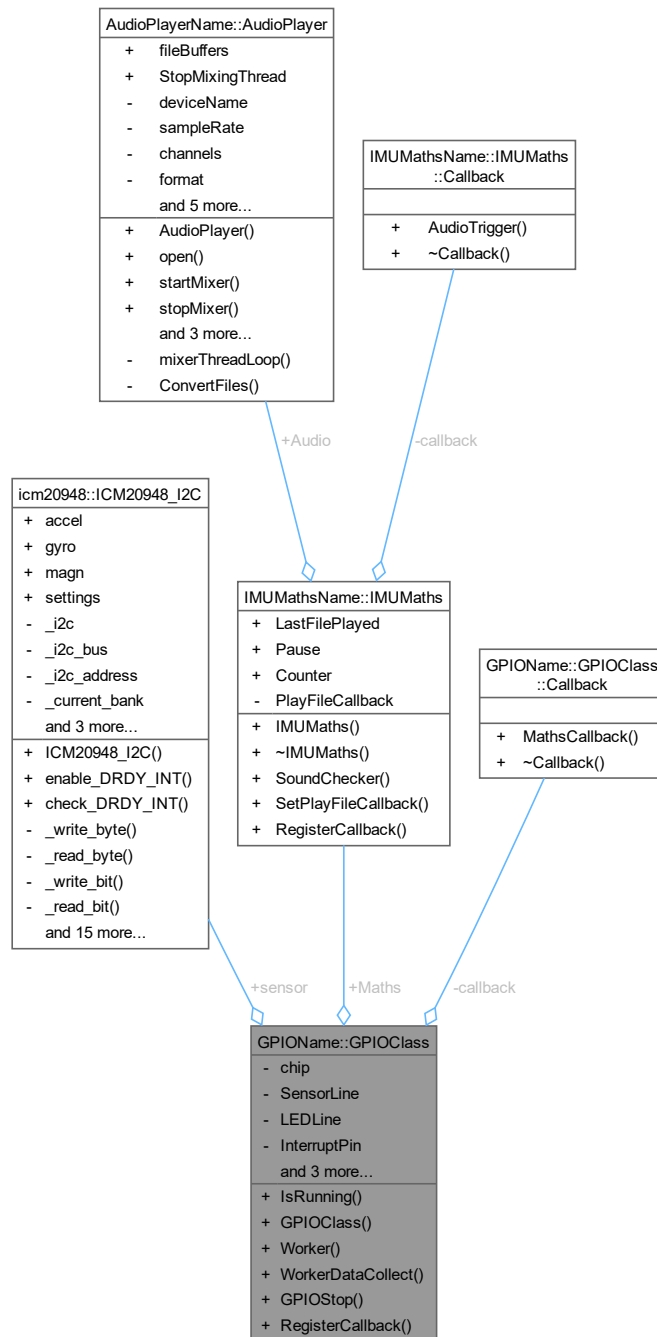
The documentation for this struct was generated from the following file:

- src/libs/IMUMaths/include/[IMUMaths.hpp](#)

## 7.6 GPIOName::GPIOClass Class Reference

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

Collaboration diagram for GPIOName::GPIOClass:



### Classes

- struct [Callback](#)  
*Callback using virtual void.*

## Public Member Functions

- bool [IsRunning](#) () const
- [GPIOClass](#) (const char \*chipName, int [InterruptPin](#), [icm20948::ICM20948\\_I2C](#) &sensor, [IMUMathsName::IMUMaths](#) &Maths)  
*Constructor for [GPIOClass](#).*
- void [Worker](#) ()  
*Event driven worker reading data when HIGH seen on GPIO.*
- void [WorkerDataCollect](#) ()  
*Event driven worker reading data when HIGH seen on GPIO and records data to a CSV.*
- void [GPIOStop](#) ()  
*Changes a boolean to end the worker.*
- void [RegisterCallback](#) ([Callback](#) \*cb)  
*Registers a callback.*

## Public Attributes

- [icm20948::ICM20948\\_I2C](#) & sensor
- [IMUMathsName::IMUMaths](#) & Maths

## Private Attributes

- [gpiod\\_chip](#) \* chip
- [gpiod\\_line](#) \* [SensorLine](#)
- [gpiod\\_line](#) \* [LEDLine](#)
- int [InterruptPin](#)
- int [Counter](#)
- bool [Pause](#) = true
- 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,
    IMUMathsName::IMUMaths & Maths)
```

Constructor for [GPIOClass](#).

#### Parameters

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

See also

[icm20948::ICM20948\\_I2C](#)

## Parameters

<i>Maths</i>	access to IMUMaths objects
--------------	----------------------------

## See also

[IMUMathsName::IMUMaths](#)

## 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 IsRunning()

```
bool GPIOName::GPIOClass::IsRunning () const [inline]
```

### 7.6.2.3 RegisterCallback()

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

Registers a callback.

## Parameters

<i>cb</i>	callback to register
-----------	----------------------

### 7.6.2.4 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.

### 7.6.2.5 WorkerDataCollect()

```
void GPIOName::GPIOClass::WorkerDataCollect ()
```

Event driven worker reading data when HIGH seen on GPIO and records data to a CSV.

The function begins by initialising a csv file named by the user. This function then uses blocking interrupts controlled by a GPIO pin. Once this GPIO pin reads HIGH the function reads the data registers using the ReadAccel() callback from the IMU's driver and appends this data into the opened CSV file.



## 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 LEDLine

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

### 7.6.3.6 Maths

```
IMUMathsName::IMUMaths& GPIOName::GPIOClass::Maths
```

### 7.6.3.7 Pause

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

### 7.6.3.8 running

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

### 7.6.3.9 sensor

```
icm20948::ICM20948_I2C& GPIOName::GPIOClass::sensor
```

### 7.6.3.10 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:

icm20948::ICM20948_I2C
<ul style="list-style-type: none"> <li>+ accel</li> <li>+ gyro</li> <li>+ magn</li> <li>+ settings</li> <li>- _i2c</li> <li>- _i2c_bus</li> <li>- _i2c_address</li> <li>- _current_bank</li> <li>and 3 more...</li> </ul>
<ul style="list-style-type: none"> <li>+ ICM20948_I2C()</li> <li>+ enable_DRDY_INT()</li> <li>+ check_DRDY_INT()</li> <li>- _write_byte()</li> <li>- _read_byte()</li> <li>- _write_bit()</li> <li>- _read_bit()</li> <li>and 15 more...</li> </ul>

### 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**

<i>i2c_bus</i>	The I2C bus number to which the sensor is connected.
<i>i2c_address</i>	The I2C address of the sensor (default is ICM20948_I2C_ADDR).
<i>settings</i>	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 receives 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 occurred

#### 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 successfully

## 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
```

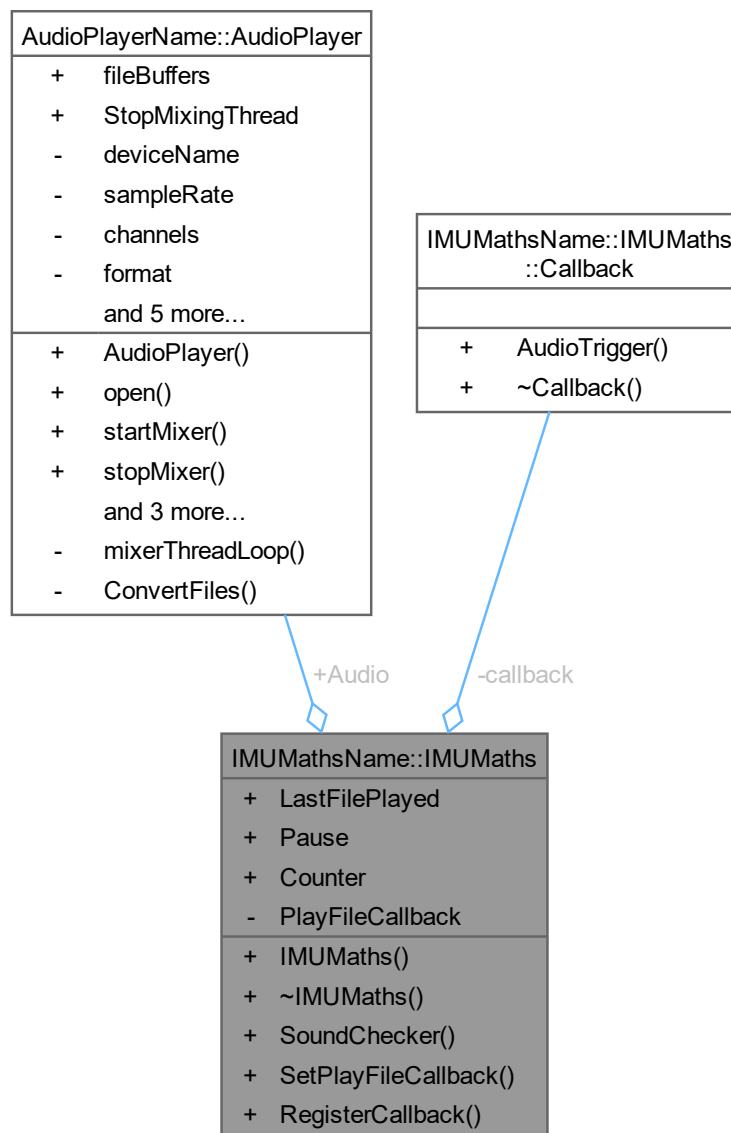
The documentation for this class was generated from the following file:

- [src/libs/I2C/include/icm20948\\_i2c.hpp](#)

## 7.8 IMUMathsName::IMUMaths Class Reference

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

Collaboration diagram for IMUMathsName::IMUMaths:





## Classes

- struct [Callback](#)  
*Callback using virtual void.*

## Public Member Functions

- [IMUMaths](#) ([AudioPlayerName::AudioPlayer](#) &[Audio](#))  
*Constructs an object with access to the audio player.*
- [~IMUMaths](#) ()  
*Destructor.*
- void [SoundChecker](#) (float X, float Y, float Z)  
*It measures each axis and sees if it falls within desired thresholds.*
- void [SetPlayFileCallback](#) (const std::function< void(const std::string &);> &cb)  
*Sets the callback.*
- void [RegisterCallback](#) ([Callback](#) \*cb)  
*Registers a callback.*

## Public Attributes

- [AudioPlayerName::AudioPlayer](#) & [Audio](#)
- int [LastFilePlayed](#)
- bool [Pause](#) = false
- int [Counter](#) = 0

## Private Attributes

- [Callback](#) \* [callback](#) = nullptr
- std::function< void(const std::string &);> [PlayFileCallback](#)

## 7.8.1 Constructor & Destructor Documentation

### 7.8.1.1 IMUMaths()

```
IMUMathsName::IMUMaths::IMUMaths (
    AudioPlayerName::AudioPlayer & Audio)
```

Constructs an object with access to the audio player.

#### Parameters

<a href="#">Audio</a>	used for playback
-----------------------	-------------------

#### See also

[AudioPlayerName::AudioPlayer](#)

### 7.8.1.2 ~IMUMaths()

```
IMUMathsName::IMUMaths::~~IMUMaths ()
```

Destructor.

## 7.8.2 Member Function Documentation

### 7.8.2.1 RegisterCallback()

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

Registers a callback.

#### Parameters

<i>cb</i>	callback to register
-----------	----------------------

### 7.8.2.2 SetPlayFileCallback()

```
void IMUMathsName::IMUMaths::SetPlayFileCallback (
    const std::function< void(const std::string &)> & cb)
```

Sets the callback.

It registers a callback via the function input

#### Parameters

<i>cb</i>	
-----------	--

### 7.8.2.3 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

<i>X</i>	acceleration along the x-axis
<i>Y</i>	acceleration along the Y-axis
<i>Z</i>	acceleration along the Z-axis

## 7.8.3 Member Data Documentation

### 7.8.3.1 Audio

```
AudioPlayerName::AudioPlayer& IMUMathsName::IMUMaths::Audio
```

### 7.8.3.2 callback

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

### 7.8.3.3 Counter

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

### 7.8.3.4 LastFilePlayed

```
int IMUMathsName::IMUMaths::LastFilePlayed
```

### 7.8.3.5 Pause

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

### 7.8.3.6 PlayFileCallback

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

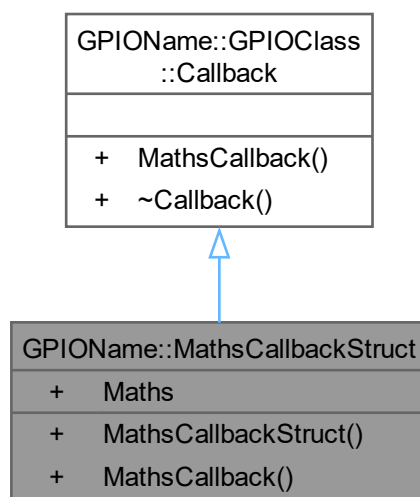
The documentation for this class was generated from the following file:

- [src/libs/IMUMaths/include/IMUMaths.hpp](#)

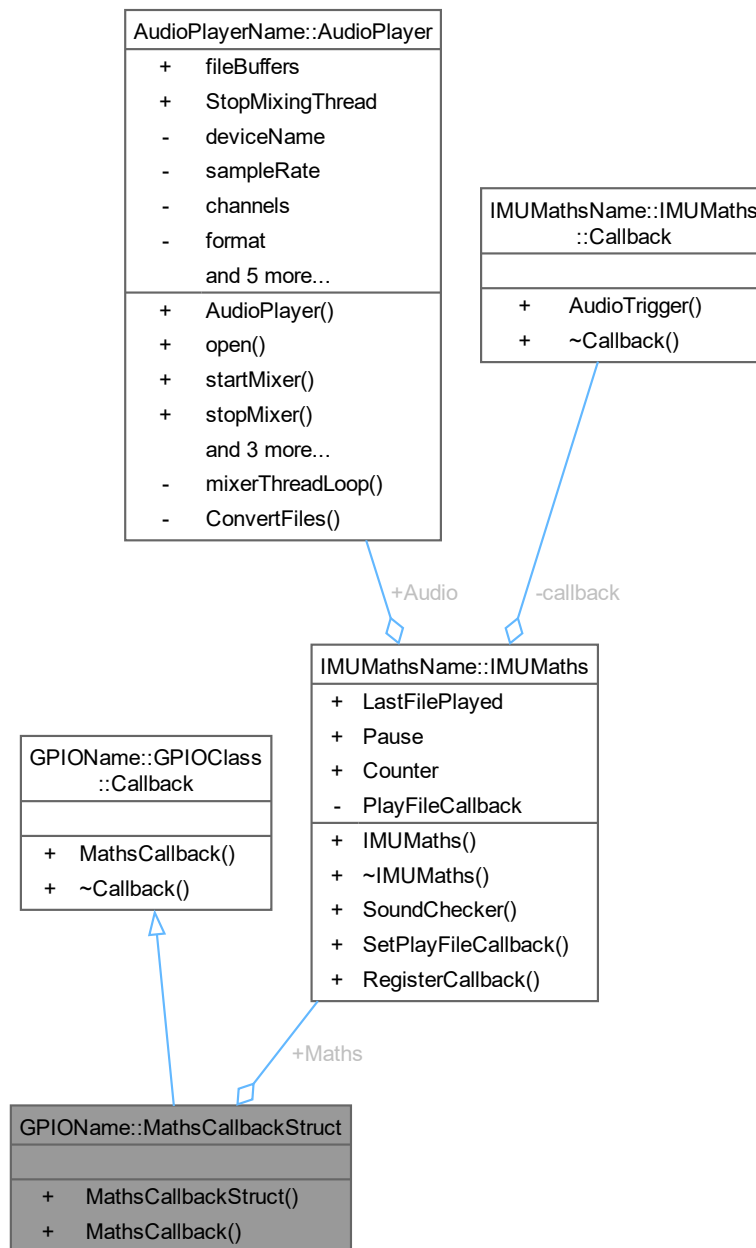
## 7.9 GPIOName::MathsCallbackStruct Struct Reference

```
#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

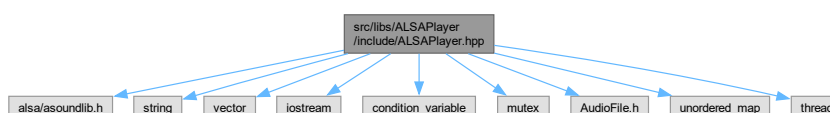
# File Documentation

### 8.1 README.md File Reference

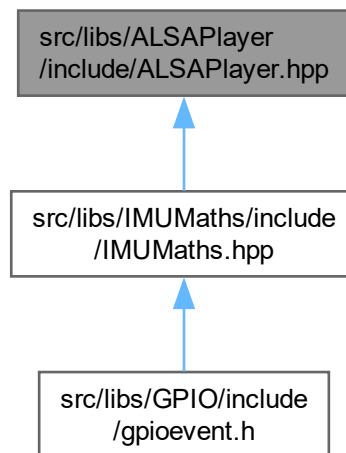
### 8.2 src/libs/ALSAPlayer/include/ALSAPlayer.hpp File Reference

```
#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:



## Classes

- class [AudioPlayerName::AudioPlayer](#)
- struct [AudioPlayerName::AudioPlayer::ActiveSound](#)

## Namespaces

- namespace [AudioPlayerName](#)

## 8.3 ALSAPlayer.hpp

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

```

00001
00002 #ifndef ALSAPLAYER_H
00003 #define ALSAPLAYER_H
00004
00005 #include <alsa/asoundlib.h>
00006 #include <string>
00007 #include <vector>
00008 #include <iostream>
00009 #include <condition_variable>
00010 #include <mutex>
00011 #include "AudioFile.h"
00012 #include <unordered_map>
00013 #include <thread>
00014
00015
00016 namespace AudioPlayerName{
00017     class AudioPlayer{
00018     public:
00019         std::unordered_map<std::string, std::vector<int32_t> > fileBuffers;
00020
00021         bool StopMixingThread = false;
00022
00023         AudioPlayer(const std::string& device="default",
00024             unsigned int rate = 44100,

```



```

00037         unsigned int ch = 2,
00038         snd_pcm_format_t fmt = SND_PCM_FORMAT_S16_LE,
00039         snd_pcm_uframes_t frames = 256,
00040         const std::vector<std::string>& filesToConvert =
00041         {"src/libs/ALSAPlayer/include/CrashCymbal.wav",
00042         "src/libs/ALSAPlayer/include/HighTom.wav",
00043         "src/libs/ALSAPlayer/include/SnareDrum.wav"}
00044         : deviceName(device), sampleRate(rate), channels(ch),
00045         format(fmt), framesPerPeriod(frames), handle(nullptr)
00046         {
00047             if (!filesToConvert.empty()) {
00048                 ConvertFiles(filesToConvert);
00049             }
00050         }
00061     bool open() {
00062         int rc = snd_pcm_open(&handle, deviceName.c_str(), SND_PCM_STREAM_PLAYBACK, 0);
00063         if (rc < 0) {
00064             std::cerr << "Unable to open PCM devices: " << snd_strerror(rc) << std::endl;
00065             return false;
00066         }
00067         snd_pcm_hw_params_t* params;
00068         snd_pcm_hw_params_alloca(&params);
00069         snd_pcm_hw_params_any(handle, params);
00070         snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
00071         snd_pcm_hw_params_set_format(handle, params, format);
00072         snd_pcm_hw_params_set_channels(handle, params, channels);
00073
00074         unsigned int rate_near = sampleRate;
00075         snd_pcm_hw_params_set_rate_near(handle, params, &rate_near, 0);
00076
00077         rc = snd_pcm_hw_params_set_period_size_near(handle, params, &framesPerPeriod, 0);
00078         if (rc < 0) {
00079             std::cerr << "Unable to set HW parameters: " << snd_strerror(rc) << std::endl;
00080             return false;
00081         }
00082         snd_pcm_uframes_t bufferSize = framesPerPeriod * 4;
00083         rc = snd_pcm_hw_params_set_buffer_size_near(handle, params, &bufferSize);
00084         if (rc < 0) {
00085             std::cerr << "Unable to set buffer size: " << snd_strerror(rc) << std::endl;
00086             return false;
00087         }
00088
00089         rc = snd_pcm_hw_params(handle, params);
00090         if (rc < 0) {
00091             std::cerr << "Unable to set HW parameters: " << snd_strerror(rc) << std::endl;
00092             return false;
00093         }
00094
00095         // Verify the final chosen period size and buffer size
00096         snd_pcm_hw_params_get_period_size(params, &framesPerPeriod, 0);
00097         snd_pcm_hw_params_get_buffer_size(params, &bufferSize);
00098         std::cout << "[DEBUG] Final period size: " << framesPerPeriod << std::endl;
00099         std::cout << "[DEBUG] Final buffer size: " << bufferSize << std::endl;
00100
00101         return true;
00102     }
00103
00104 void startMixer() {
00105     if (!handle) {
00106         std::cerr << "ALSA device is not open. Call open() first." << std::endl;
00107         return;
00108     }
00109     StopMixingThread = false;
00110     mixThread = std::thread(&AudioPlayer::mixerThreadLoop, this);
00111 }
00112
00113 void stopMixer() {
00114     StopMixingThread = true;
00115     if (mixThread.joinable()) {
00116         mixThread.join();
00117     }
00118 }
00119
00120 bool addSoundToMixer(const std::string& fileKey) {
00121     std::lock_guard<std::mutex> lock(ActiveMutex);
00122
00123     // Check if file buffer exists
00124     auto it = fileBuffers.find(fileKey);
00125     if (it == fileBuffers.end()) {
00126         std::cerr << "Audio buffer not found for file: " << fileKey << std::endl;
00127         return false;
00128     }
00129 }

```

```

00147         }
00148
00149         // Create a new ActiveSound
00150         ActiveSound newSound;
00151
00152         // pointer to the file's buffer
00153         newSound.buffer = &it->second;
00154         newSound.position = 0;
00155
00156         // Add it to active sounds
00157         ActiveSounds.push_back(newSound);
00158         return true;
00159     }
00160
00161 void close() {
00162     stopMixer();
00163     if (handle) {
00164         snd_pcm_drop(handle);
00165         snd_pcm_close(handle);
00166         handle = nullptr;
00167     }
00168 }
00169
00170 ~AudioPlayer() {
00171     close();
00172 }
00173
00174 private:
00175     std::string deviceName;
00176     unsigned int sampleRate;
00177     unsigned int channels;
00178     snd_pcm_format_t format;
00179     snd_pcm_uframes_t framesPerPeriod;
00180     snd_pcm_t* handle;
00181
00182     std::thread mixThread;
00183
00184     struct ActiveSound {
00185         std::vector<int32_t>* buffer;
00186         size_t position;
00187     };
00188
00189     std::vector<ActiveSound> ActiveSounds;
00190     std::mutex ActiveMutex;
00191
00192 void mixerThreadLoop() {
00193     // Allocate a buffer for one period of audio
00194     const size_t periodSizeSamples = framesPerPeriod * channels;
00195     std::vector<int32_t> mixBuffer(periodSizeSamples, 0);
00196
00197     while (!StopMixingThread) {
00198         // Clear the mix buffer each iteration
00199         std::fill(mixBuffer.begin(), mixBuffer.end(), 0);
00200
00201         {
00202             // Locks the active list of sounds
00203             std::lock_guard<std::mutex> lock(ActiveMutex);
00204
00205             // Mixes all of the activesounds and removes those that have finished
00206             for (auto it = ActiveSounds.begin(); it != ActiveSounds.end(); ) {
00207                 ActiveSound& sound = *it;
00208                 const size_t totalFrames = sound.buffer->size() / channels;
00209                 size_t framesLeft = totalFrames - sound.position;
00210
00211                 // Calculate how many frames are left to be mixed
00212                 size_t framesToMix = std::min<size_t>(framesPerPeriod, framesLeft);
00213
00214                 // Mix the audio data from this sound into the buffer
00215                 for (size_t f = 0; f < framesToMix; ++f) {
00216                     for (unsigned int c = 0; c < channels; ++c) {
00217                         // Source index in the file buffer
00218                         size_t srcIndex = (sound.position + f) * channels + c;
00219                         // Destination index in the mix buffer
00220                         size_t dstIndex = f * channels + c;
00221
00222                         // Add up the sample
00223                         mixBuffer[dstIndex] += (*sound.buffer)[srcIndex];
00224                     }
00225                 }
00226
00227                 // Advance playback position
00228                 sound.position += framesToMix;
00229
00230                 // If a sound has finished, remove it
00231                 if (sound.position >= totalFrames) {
00232                     it = ActiveSounds.erase(it);
00233                 }
00234             }
00235         }
00236     }

```

```

00240             } else {
00241                 ++it;
00242             }
00243         }
00244     }
00245
00246     // Write the mixed buffer to ALSA
00247     int rc = snd_pcm_writei(handle, mixBuffer.data(), framesPerPeriod);
00248     if (rc == -EPIPE) {
00249         std::cerr << "Underrun occurred\n";
00250         snd_pcm_prepare(handle);
00251     } else if (rc < 0) {
00252         std::cerr << "Error from writei: " << snd_strerror(rc) << std::endl;
00253     }
00254 }
00255 }
00256
00257
00258 void ConvertFiles(const std::vector<std::string>& filePaths) {
00259     std::vector<int32_t> result;
00260
00261     for (const auto& path : filePaths) {
00262         AudioFile<int32_t> file;
00263         if (!file.load(path)) {
00264             std::cerr << "Error loading file: " << path << std::endl;
00265             continue;
00266         }
00267
00268         int fileChannels = file.getNumChannels();
00269         int ChannelSamples = file.getNumSamplesPerChannel();
00270
00271         std::vector<int32_t> interleaved;
00272         interleaved.reserve(ChannelSamples * fileChannels);
00273         for (int i=0; i < ChannelSamples; ++i){
00274             for (int ch = 0; ch < fileChannels; ++ch){
00275                 interleaved.push_back(file.samples[ch][i]);
00276             }
00277         }
00278
00279         }
00280         fileBuffers[path] = std::move(interleaved);
00281     }
00282 }
00283 }
00284 }
00285
00286 };
00287 };
00288 }
00289
00290
00291
00292 #endif

```

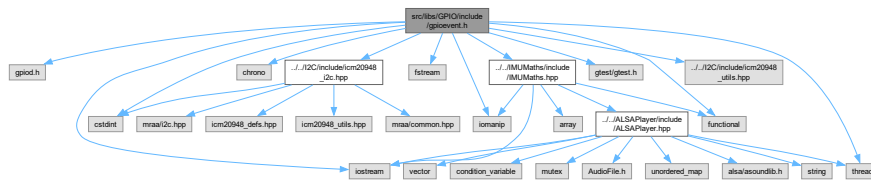
## 8.4 src/libs/GPIO/include/gpioevent.h File Reference

```

#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](#)  
*Callback using virtual void.*
- struct [GPIOName::MathsCallbackStruct](#)

## Namespaces

- namespace [GPIOName](#)

## Typedefs

- typedef void(\* [GPIOName::GPIOCallback](#)) (void \*context, float, float, float)

## 8.5 gpioevent.h

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

```

00001 #ifndef GPIOEVENT_H
00002 #define GPIOEVENT_H
00003
00004
00005 #include <gpiod.h>
00006 #include <iostream>
00007 #include <thread>
00008 #include <chrono>
00009 #include <iomanip>
00010 #include <fstream>
00011 #include <cstdint>
00012 #include <functional>
00013 #include <gtest/gtest.h>
00014
00015 #include "../I2C/include/icm20948_i2c.hpp"
00016 #include "../I2C/include/icm20948_utils.hpp"
00017
00018 #include "../IMUMaths/include/IMUMaths.hpp"
00019
00020
00021 namespace GPIOName {
00022
00023     typedef void (*GPIOCallback) (void* context, float, float, float);
00024     class GPIOClass {
00025     public:
00026         icm20948::ICM20948_I2C& sensor;
00027         IMUMathsName::IMUMaths& Maths;
00028
00029         // GPIOCallback callback;
00030         // void* CallbackFunction;
00031
00032         // function added for testing
00033         bool IsRunning() const {

```

```

00034         return running.load();
00035     }
00036
00047     GPIOClass(const char* chipName, int InterruptPin,
00048               icm20948::ICM20948_I2C& sensor, IMUMathsName::IMUMaths& Maths);
00049
00058     void Worker();
00059
00070     void WorkerDataCollect();
00071
00075     void GPIOStop();
00076
00080     struct Callback{
00081         virtual void MathsCallback(float X, float Y, float Z) = 0;
00082         virtual ~Callback(){};
00083     };
00084
00090     void RegisterCallback(Callback* cb){
00091         callback = cb;
00092     }
00093
00094     // void SetCallback(GPIOCallback cb, void* context);
00095     // static void IMUMathsCallback(void* context, float X, float Y, float Z){
00096     //     IMUMathsName::IMUMaths* maths = static_cast<IMUMathsName::IMUMaths*>(context);
00097     //     maths->SoundChecker(X,Y,Z);
00098     // }
00099
00100     //Testing private stuff
00101     #ifdef UNIT_TEST
00102         bool GetRunning() const {
00103             return running.load();
00104         }
00105
00106         bool HasCallback() const {
00107             return callback != nullptr;
00108         }
00109
00110         Callback* GetCallback() const {
00111             return callback;
00112         }
00113
00114     #endif
00115
00116     private:
00117         gpiod_chip* chip;
00118         gpiod_line* SensorLine;
00119         gpiod_line* LEDLine;
00120         int InterruptPin;
00121         int Counter;
00122         bool Pause = true;
00123
00124         std::atomic<bool> running{true};
00125         Callback* callback = nullptr;
00126
00127     };
00128
00129     struct MathsCallbackStruct : GPIOName::GPIOClass::Callback{
00130         IMUMathsName::IMUMaths& Maths;
00131
00132         MathsCallbackStruct(IMUMathsName::IMUMaths& maths) : Maths(maths) {}
00133
00134         virtual void MathsCallback(float X, float Y, float Z) override {
00135             Maths.SoundChecker(X, Y, Z);
00136         }
00137     };
00138 }
00139
00140
00141
00142 #endif

```

## 8.6 src/libs/I2C/include/icm20948\_i2c.hpp File Reference

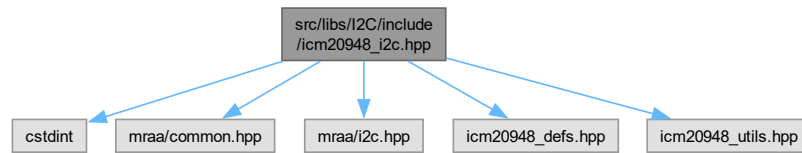
```

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

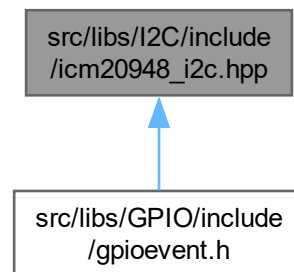
```

```
#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.](#)

```

00001 #ifndef ICM20948_I2C_HPP
00002 #define ICM20948_I2C_HPP
00003
00004 #include <stdint>
00005
00006 #include "mraa/common.hpp"
00007 #include "mraa/i2c.hpp"
00008
00009 #include "icm20948_defs.hpp"
00010 #include "icm20948_utils.hpp"
00011
00012 namespace icm20948
00013 {

```

```

00014     class ICM20948_I2C
00015     {
00016     private:
00017         mraa::I2c _i2c;
00018         unsigned _i2c_bus, _i2c_address;
00019         uint8_t _current_bank;
00020         float _accel_scale_factor, _gyro_scale_factor, _magn_scale_factor;
00021
00022         bool _write_byte(const uint8_t bank, const uint8_t reg, const uint8_t byte);
00023         bool _read_byte(const uint8_t bank, const uint8_t reg, uint8_t &byte);
00024         bool _write_bit(const uint8_t bank, const uint8_t reg, const uint8_t bit_pos, const bool
bit);
00025         bool _read_bit(const uint8_t bank, const uint8_t reg, const uint8_t bit_pos, bool &bit);
00026         bool _read_block_bytes(const uint8_t bank, const uint8_t start_reg, uint8_t *bytes, const
int length);
00027         bool _write_mag_byte(const uint8_t mag_reg, const uint8_t byte);
00028         bool _read_mag_byte(const uint8_t mag_reg, uint8_t &byte);
00029         bool _read_int_byte(const uint8_t bank, const uint8_t reg, uint8_t &byte);
00030
00031         bool _set_bank(uint8_t bank);
00032         bool _set_accel_sample_rate_div();
00033         bool _set_accel_range_dlpf();
00034         bool _set_gyro_sample_rate_div();
00035         bool _set_gyro_range_dlpf();
00036
00037         bool _magnetometer_init();
00038         bool _magnetometer_enable();
00039         bool _magnetometer_set_mode();
00040         bool _magnetometer_configured();
00041         bool _magnetometer_set_readout();
00042
00043         bool _chip_i2c_master_reset();
00044
00045     public:
00046         // Contains linear acceleration in m/s^2
00047         float accel[3];
00048         // Contains angular velocities in rad/s
00049         float gyro[3];
00050         // Contains magnetic field strength in uTesla
00051         float magn[3];
00052
00053         // Sensor settings
00054         icm20948::settings settings;
00055
00065         ICM20948_I2C(unsigned i2c_bus, unsigned i2c_address = ICM20948_I2C_ADDR,
icm20948::settings
00066             = icm20948::settings());
00067
00069
00085         bool init();
00086
00100         bool reset();
00101
00112         bool wake();
00113
00129         bool set_settings();
00130
00144         bool read_accel_gyro();
00145
00157         bool read_magn();
00158
00160
00171         bool enable_DRDY_INT();
00172
00182         bool check_DRDY_INT();
00183     };
00184 }
00185
00186 #endif

```

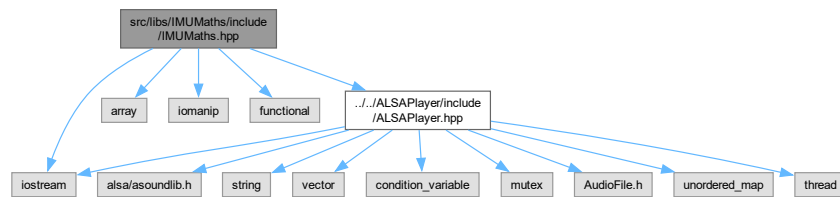
## 8.8 src/libs/IMUMaths/include/IMUMaths.hpp File Reference

```

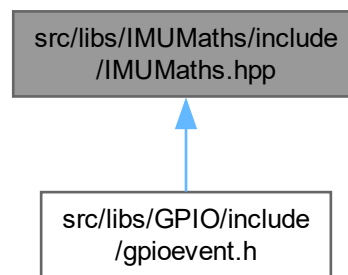
#include <iostream>
#include <array>
#include <iomanip>
#include <functional>
#include "../ALSAPlayer/include/ALSAPlayer.hpp"

```

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](#)  
*Callback using virtual void.*
- struct [IMUMathsName::AudioCallback](#)

## Namespaces

- namespace [IMUMathsName](#)

## 8.9 IMUMaths.hpp

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

```

00001 #ifndef IMUMATHS_H
00002 #define IMUMATHS_H
00003
00004
00005 #include <iostream>
00006 #include <array>
00007 #include <iomanip>
00008 #include <functional>
00009

```



```

00010 #include "../ALSAPlayer/include/ALSAPlayer.hpp"
00011
00012
00013 namespace IMUMathsName {
00014     class IMUMaths{
00015
00016     public:
00017         AudioPlayerName::AudioPlayer &Audio;
00018
00019         IMUMaths(AudioPlayerName::AudioPlayer &Audio);
00020
00021         ~IMUMaths();
00022
00023         // For debugging: Identifier of the last audio file played
00024
00025         int LastFilePlayed;
00026
00027         void SoundChecker(float X, float Y, float Z);
00028
00029         void SetPlayFileCallback(const std::function<void(const std::string*)>& cb);
00030
00031         // Pauses
00032         bool Pause = false;
00033
00034         // Counter variable
00035         int Counter = 0;
00036
00037         struct Callback{
00038             virtual void AudioTrigger(const std::string& FilePath) = 0;
00039             virtual ~Callback(){};
00040         };
00041
00042         void RegisterCallback(Callback* cb){
00043             callback = cb;
00044             //std::cout << "[IMUMaths] Registered callback at address: " << callback << std::endl;
00045         }
00046
00047         //Access to private fo UNIT_TEST only
00048         #ifdef UNIT_TEST
00049
00050         bool HasCallback() const {
00051             return callback != nullptr;
00052         }
00053
00054         Callback* GetCallback() const {
00055             return callback;
00056         }
00057
00058         #endif
00059
00060     private:
00061
00062         Callback* callback = nullptr;
00063         std::function<void(const std::string*)> PlayFileCallback;
00064
00065     };
00066
00067     struct AudioCallback : IMUMathsName::IMUMaths::Callback{
00068         AudioPlayerName::AudioPlayer& Audio;
00069
00070         AudioCallback(AudioPlayerName::AudioPlayer& audio) : Audio(audio) {}
00071
00072         virtual void AudioTrigger(const std::string& FilePath){
00073             std::thread([this,FilePath]{
00074                 Audio.addSoundToMixer(FilePath);
00075             }).detach();
00076         }
00077     };
00078 }
00079
00080 #endif

```

