

SnAirBeats

1.0

Generated by Doxygen 1.13.2

1 SnAirBeats	1
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
2 Namespace Index	5
2.1 Namespace List	5
3 Hierarchical Index	7
3.1 Class Hierarchy	7
4 Class Index	9
4.1 Class List	9
5 File Index	11
5.1 File List	11
6 Namespace Documentation	13
6.1 AudioPlayerName Namespace Reference	13
6.2 GPIOName Namespace Reference	13
6.3 icm20948 Namespace Reference	13
6.4 IMUMathsName Namespace Reference	13
7 Class Documentation	15
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()	22
7.3.2.3 ConvertFiles()	22
7.3.2.4 mixerThreadLoop()	22
7.3.2.5 open()	23
7.3.2.6 startMixer()	23
7.3.2.7 stopMixer()	24
7.3.3 Member Data Documentation	24
7.3.3.1 ActiveMutex	24
7.3.3.2 ActiveSounds	24
7.3.3.3 channels	24
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	25
7.3.3.9 mixThread	25
7.3.3.10 sampleRate	25
7.3.3.11 StopMixingThread	25
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	27
7.5.1 Detailed Description	28
7.5.2 Constructor & Destructor Documentation	28
7.5.2.1 ~Callback()	28
7.5.3 Member Function Documentation	28
7.5.3.1 AudioTrigger()	28
7.6 GPIOName::GPIOClass Class Reference	28

7.6.1 Constructor & Destructor Documentation	30
7.6.1.1 GPIOClass()	30
7.6.2 Member Function Documentation	30
7.6.2.1 GPIOStop()	30
7.6.2.2 RegisterCallback()	30
7.6.2.3 Worker()	31
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 Pause	31
7.6.3.6 running	31
7.6.3.7 sensor	31
7.6.3.8 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 <code>_current_bank</code>	37
7.7.3.3 <code>_gyro_scale_factor</code>	37
7.7.3.4 <code>_i2c</code>	37
7.7.3.5 <code>_i2c_address</code>	37
7.7.3.6 <code>_i2c_bus</code>	37
7.7.3.7 <code>_magn_scale_factor</code>	37
7.7.3.8 <code>accel</code>	37
7.7.3.9 <code>gyro</code>	37
7.7.3.10 <code>magn</code>	37
7.7.3.11 <code>settings</code>	38
7.8 IMUMathsName::IMUMaths Class Reference	38
7.8.1 Member Function Documentation	39
7.8.1.1 <code>RegisterCallback()</code>	39
7.8.1.2 <code>SoundChecker()</code>	39
7.8.2 Member Data Documentation	39
7.8.2.1 <code>callback</code>	39
7.8.2.2 <code>Counter</code>	40
7.8.2.3 <code>LastFilePlayed</code>	40
7.8.2.4 <code>Pause</code>	40
7.8.2.5 <code>PlayFileCallback</code>	40
7.9 GPIOName::MathsCallbackStruct Struct Reference	40
7.9.1 Constructor & Destructor Documentation	42
7.9.1.1 <code>MathsCallbackStruct()</code>	42
7.9.2 Member Function Documentation	42
7.9.2.1 <code>MathsCallback()</code>	42
7.9.3 Member Data Documentation	42
7.9.3.1 <code>Maths</code>	42
8 File Documentation	43
8.1 README.md File Reference	43
8.2 <code>src/libs/ALSAPlayer/include/ALSAPlayer.hpp</code> File Reference	43
8.3 <code>ALSAPlayer.hpp</code>	44
8.4 <code>src/libs/GPIO/include/gpioevent.h</code> File Reference	47
8.5 <code>gpioevent.h</code>	48
8.6 <code>src/libs/I2C/include/icm20948_i2c.hpp</code> File Reference	49
8.7 <code>icm20948_i2c.hpp</code>	50
8.8 <code>src/libs/IMUMaths/include/IMUMaths.hpp</code> File Reference	51
8.9 <code>IMUMaths.hpp</code>	52

Chapter 1

SnAirBeats



1.1 SnAIRbeats

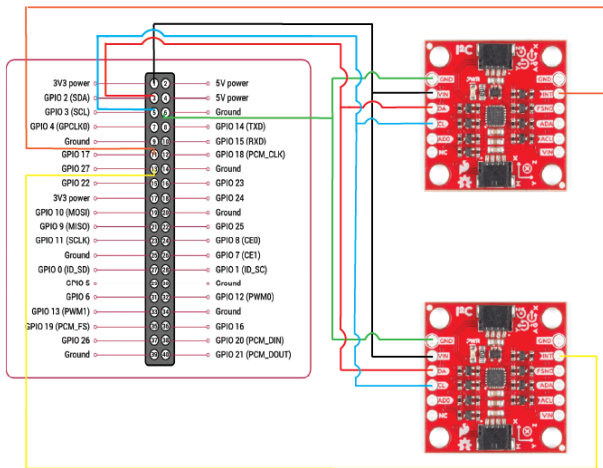
SnAirBeats is a next generation method to practice the drums, while reducing noise and space typically required to do so. The SnAirBeat set uses inertial measurement units (IMU) within the sticks to track their movement and play a corresponding drum, not requiring any physical hitting like modern electric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

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:

AudioPlayerName	13
GPIOName	13
icm20948	13
IMUMathsName	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	40
IMUMathsName::IMUMaths::Callback	27
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:

AudioPlayerName::AudioPlayer::ActiveSound	15
IMUMathsName::AudioCallback	16
AudioPlayerName::AudioPlayer	18
GPIOName::GPIOClass::Callback	
Empty callback to later be filled. Includes destructor	25
IMUMathsName::IMUMaths::Callback	
Empty callback to later be filled. Includes destructor	27
GPIOName::GPIOClass	28
icm20948::ICM20948_I2C	32
IMUMathsName::IMUMaths	38
GPIOName::MathsCallbackStruct	40

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

src/libs/ALSAPlayer/include/ ALSAPlayer.hpp	43
src/libs/GPIO/include/ gpioevent.h	47
src/libs/I2C/include/ icm20948_i2c.hpp	49
src/libs/IMUMaths/include/ IMUMaths.hpp	51

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

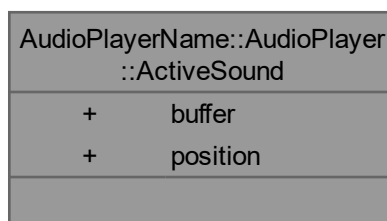
- struct [AudioCallback](#)
- class [IMUMaths](#)

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
```

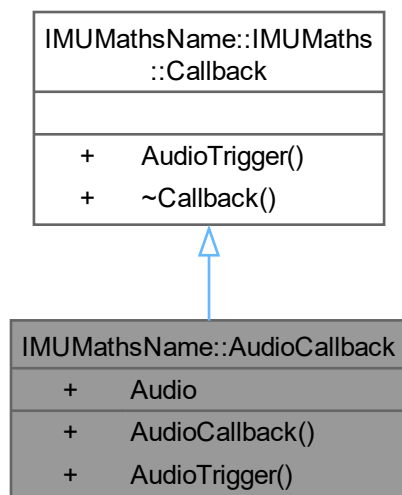
The documentation for this struct was generated from the following file:

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

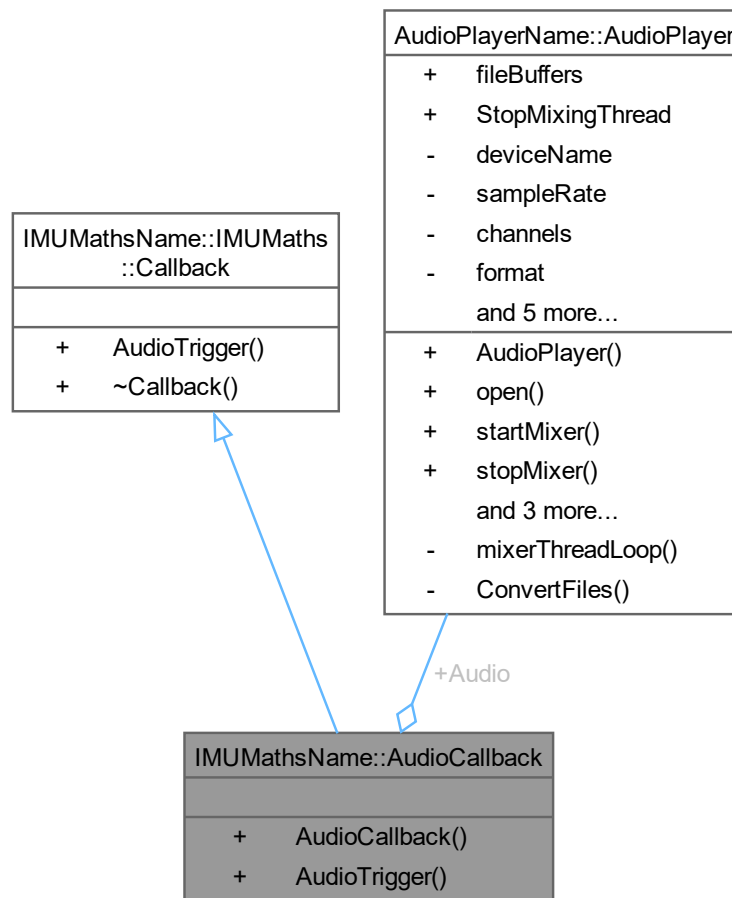
7.2 IMUMathsName::AudioCallback Struct Reference

```
#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:

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.
- 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

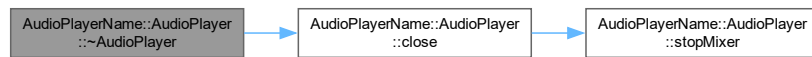
<i>device</i>	The name of the ALSA device to use.
<i>rate</i>	Sample rate in Hz.
<i>ch</i>	Number of channels.
<i>fmt</i>	Format of audio data.
<i>frames</i>	Number of frames per period.
<i>filesToConvert</i>	Sound files used.

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.
----------------	-----------------

Returns

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

7.3.2.2 close()

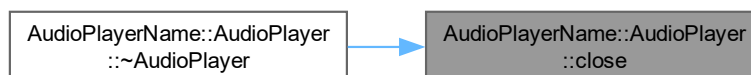
```
void AudioPlayerName::AudioPlayer::close () [inline]
```

Close PCM handle and free all associated resources.

Here is the call graph for this function:



Here is the caller graph for this function:



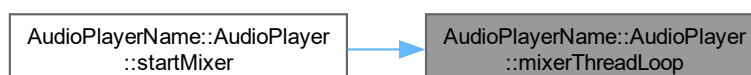
7.3.2.3 ConvertFiles()

```
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

Returns

Returns true if 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 if 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:

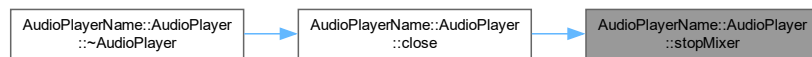


7.3.2.7 stopMixer()

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

Stop mixer thread, closes thread when called.

Here is the caller graph for this function:



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:

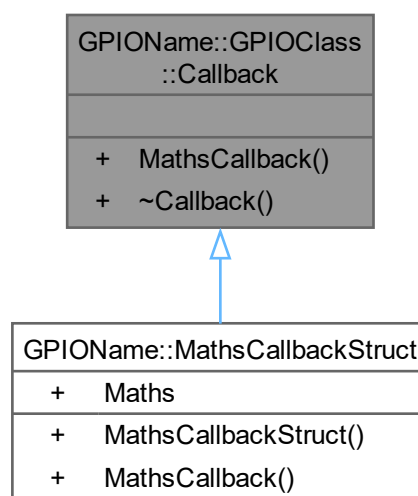
- src/libs/ALSAPlayer/include/[ALSAPlayer.hpp](#)

7.4 GPIOName::GPIOClass::Callback Struct Reference

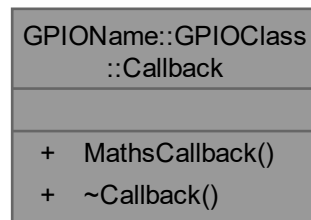
Empty callback to later be filled. Includes destructor.

```
#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:

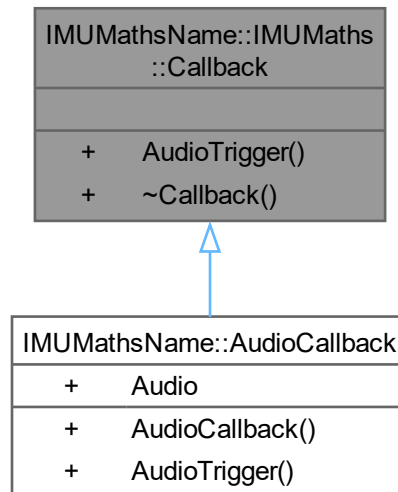
- [src/libs/GPIO/include/gpioevent.h](#)

7.5 IMUMathsName::IMUMaths::Callback Struct Reference

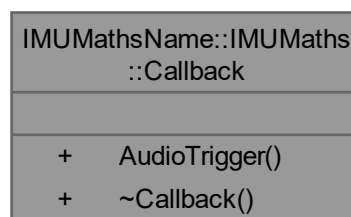
Empty callback to later be filled. Includes destructor.

```
#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](#).

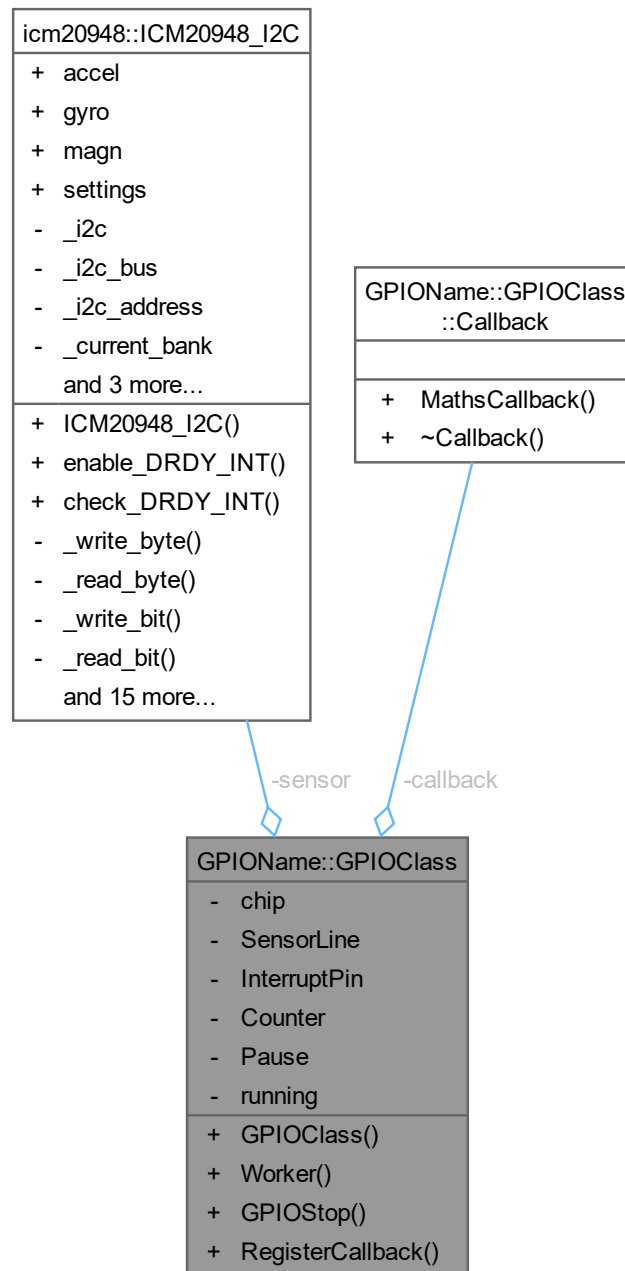
The documentation for this struct was generated from the following file:

- [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

<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](#)

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

<i>cb</i>	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:

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

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
```

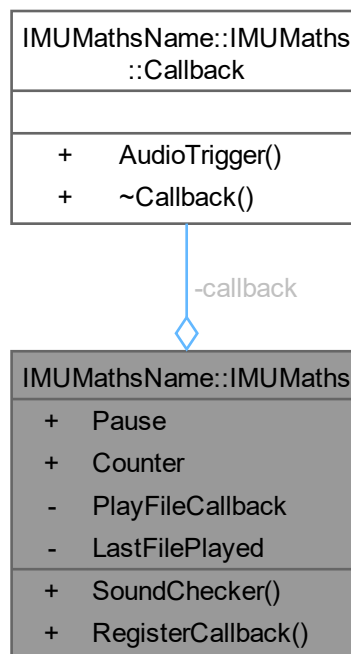
The documentation for this class was generated from the following file:

- [src/libs/I2C/include/icm20948_i2c.hpp](#)

7.8 IMUMathsName::IMUMaths Class Reference

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

Collaboration diagram for IMUMathsName::IMUMaths:



Classes

- struct [Callback](#)
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

<i>cb</i>	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 thresholds, 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

7.8.2 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]
```

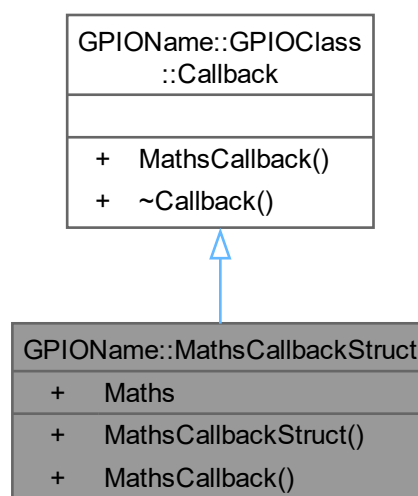
The documentation for this class was generated from the following file:

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

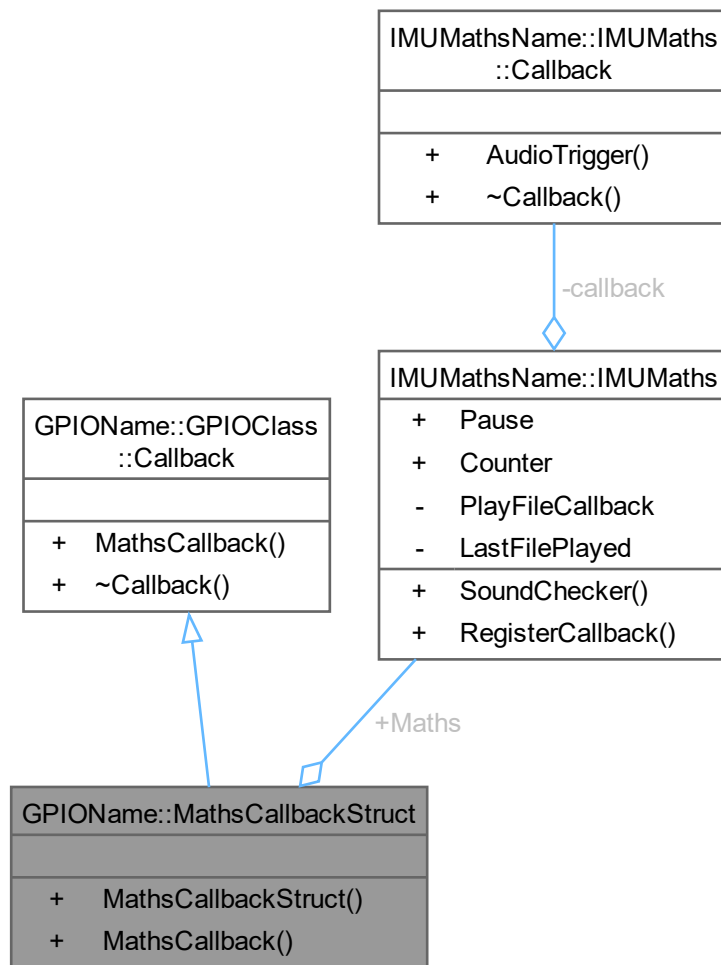
7.9 GPIOName::MathsCallbackStruct Struct Reference

```
#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

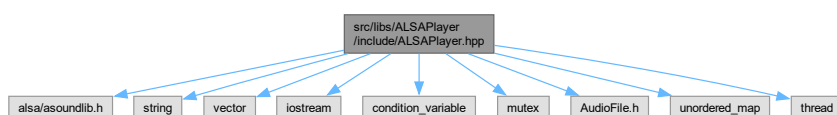
File Documentation

8.1 README.md File Reference

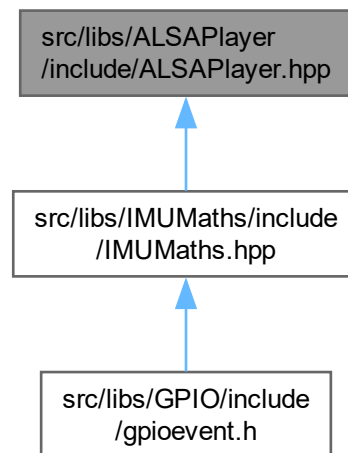
8.2 src/libs/ALSAPlayer/include/ALSAPlayer.hpp File Reference

```
#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:



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         }
00063     bool open() {
00064         int rc = snd_pcm_open(&handle, deviceName.c_str(), SND_PCM_STREAM_PLAYBACK, 0);
00065         if (rc < 0) {
00066             std::cerr << "Unable to open PCM devices: " << snd_strerror(rc) << std::endl;
00067             return false;
00068         }
00069
00070         snd_pcm_hw_params_t* params;
00071         snd_pcm_hw_params_alloca(&params);
00072         snd_pcm_hw_params_any(handle, params);
00073         snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
00074         snd_pcm_hw_params_set_format(handle, params, format);
00075         snd_pcm_hw_params_set_channels(handle, params, channels);
00076
00077         unsigned int rate_near = sampleRate;
00078         snd_pcm_hw_params_set_rate_near(handle, params, &rate_near, 0);
00079
00080         rc = snd_pcm_hw_params_set_period_size_near(handle, params, &framesPerPeriod, 0);
00081         if (rc < 0) {
00082             std::cerr << "Unable to set HW parameters: " << snd_strerror(rc) << std::endl;
00083             return false;
00084         }
00085         snd_pcm_uframes_t bufferSize = framesPerPeriod * 4;
00086         rc = snd_pcm_hw_params_set_buffer_size_near(handle, params, &bufferSize);
00087         if (rc < 0) {
00088             std::cerr << "Unable to set buffer size: " << snd_strerror(rc) << std::endl;
00089             return false;
00090         }
00091
00092         rc = snd_pcm_hw_params(handle, params);
00093         if (rc < 0) {
00094             std::cerr << "Unable to set HW parameters: " << snd_strerror(rc) << std::endl;
00095             return false;
00096         }
00097
00098         // Verify the final chosen period size and buffer size
00099         snd_pcm_hw_params_get_period_size(params, &framesPerPeriod, 0);
00100         snd_pcm_hw_params_get_buffer_size(params, &bufferSize);
00101         std::cout << "[DEBUG] Final period size: " << framesPerPeriod << std::endl;
00102         std::cout << "[DEBUG] Final buffer size: " << bufferSize << std::endl;
00103
00104         return true;
00105     }
00106
00107     bool startMixer() {
00108         if (!handle) {
00109             std::cerr << "ALSA device is not open. Call open() first." << std::endl;
00110             return false;
00111         }
00112         StopMixingThread = false;
00113         mixThread = std::thread(&AudioPlayer::mixerThreadLoop, this);
00114
00115         return true;
00116     }
00127
00128     void stopMixer() {
00129         StopMixingThread = true;
00130         if (mixThread.joinable()) {
00131             mixThread.join();
00132         }
00133     }
00134
00135     bool addSoundToMixer(const std::string& fileKey) {
00136         std::lock_guard<std::mutex> lock(ActiveMutex);
00137
00138         // Check if file buffer exists
00139         auto it = fileBuffers.find(fileKey);
00140         if (it == fileBuffers.end()) {

```

```

00156         std::cerr << "Audio buffer not found for file: " << fileKey << std::endl;
00157         return false;
00158     }
00159
00160     // Create a new ActiveSound
00161     ActiveSound newSound;
00162
00163     // pointer to the file's buffer
00164     newSound.buffer = &it->second;
00165     newSound.position = 0;
00166
00167     // Add it to active sounds
00168     ActiveSounds.push_back(newSound);
00169     return true;
00170 }
00171
00172 void close() {
00173     stopMixer();
00174     if (handle) {
00175         snd_pcm_drop(handle);
00176         snd_pcm_close(handle);
00177         handle = nullptr;
00178     }
00179 }
00180
00181 ~AudioPlayer() {
00182     close();
00183 }
00184
00185 private:
00186     std::string deviceName;
00187     unsigned int sampleRate;
00188     unsigned int channels;
00189     snd_pcm_format_t format;
00190     snd_pcm_uframes_t framesPerPeriod;
00191     snd_pcm_t* handle;
00192
00193     std::thread mixThread;
00194
00195     struct ActiveSound {
00196         std::vector<int32_t>* buffer;
00197         size_t position;
00198     };
00199
00200     std::vector<ActiveSound> ActiveSounds;
00201     std::mutex ActiveMutex;
00202
00203     void mixerThreadLoop() {
00204         // Allocate a buffer for one period of audio
00205         const size_t periodSizeSamples = framesPerPeriod * channels;
00206         std::vector<int32_t> mixBuffer(periodSizeSamples, 0);
00207
00208         while (!StopMixingThread) {
00209             // Clear the mix buffer each iteration
00210             std::fill(mixBuffer.begin(), mixBuffer.end(), 0);
00211
00212             {
00213                 // Locks the active list of sounds
00214                 std::lock_guard<std::mutex> lock(ActiveMutex);
00215
00216                 // Mixes all of the active sounds and removes those that have finished
00217                 for (auto it = ActiveSounds.begin(); it != ActiveSounds.end(); ) {
00218                     ActiveSound& sound = *it;
00219                     const size_t totalFrames = sound.buffer->size() / channels;
00220                     size_t framesLeft = totalFrames - sound.position;
00221
00222                     // Calculate how many frames are left to be mixed
00223                     size_t framesToMix = std::min<size_t>(framesPerPeriod, framesLeft);
00224
00225                     // Mix the audio data from this sound into the buffer
00226                     for (size_t f = 0; f < framesToMix; ++f) {
00227                         for (unsigned int c = 0; c < channels; ++c) {
00228                             // Source index in the file buffer
00229                             size_t srcIndex = (sound.position + f) * channels + c;
00230                             // Destination index in the mix buffer
00231                             size_t dstIndex = f * channels + c;
00232
00233                             // Add up the sample
00234                             mixBuffer[dstIndex] += (*sound.buffer)[srcIndex];
00235                         }
00236                     }
00237
00238                     // Advance playback position
00239                     sound.position += framesToMix;
00240
00241                     // If a sound has finished, remove it

```

```

00249         if (sound.position >= totalFrames) {
00250             it = ActiveSounds.erase(it);
00251         } else {
00252             ++it;
00253         }
00254     }
00255 }
00256
00257 // Write the mixed buffer to ALSA
00258 int rc = snd_pcm_writei(handle, mixBuffer.data(), framesPerPeriod);
00259 if (rc == -EPIPE) {
00260     std::cerr << "Underrun occurred\n";
00261     snd_pcm_prepare(handle);
00262 } else if (rc < 0) {
00263     std::cerr << "Error from writei: " << snd_strerror(rc) << std::endl;
00264 }
00265 }
00266 }
00267
00268 void ConvertFiles(const std::vector<std::string>& filePaths) {
00269     std::vector<int32_t> result;
00270
00271     for (const auto& path : filePaths) {
00272         AudioFile<int32_t> file;
00273         if (!file.load(path)) {
00274             std::cerr << "Error loading file: " << path << std::endl;
00275             continue;
00276         }
00277
00278         int fileChannels = file.getNumChannels();
00279         int ChannelSamples = file.getNumSamplesPerChannel();
00280
00281         std::vector<int32_t> interleaved;
00282         interleaved.reserve(ChannelSamples * fileChannels);
00283         for (int i=0; i < ChannelSamples; ++i){
00284             for (int ch = 0; ch < fileChannels; ++ch){
00285                 interleaved.push_back(file.samples[ch][i]);
00286             }
00287         }
00288
00289         fileBuffers[path] = std::move(interleaved);
00290     }
00291 }
00292
00293 }
00294 }
00295 }
00296
00297 };
00298 }
00299 }
00300
00301
00302
00303 #endif

```

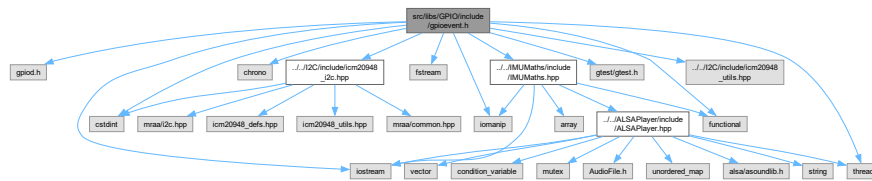
8.4 src/libs/GPIO/include/gpioevent.h File Reference

```

#include <gpiod.h>
#include <iostream>
#include <thread>
#include <chrono>
#include <iomanip>
#include <fstream>
#include <stdint>
#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.](#)

```

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     class GPIOClass {
00023     public:
00024
00025         GPIOClass(const char* chipName, int InterruptPin,
00026                 icm20948::ICM20948_I2C& sensor);
00027
00028         void Worker();
00029
00030         void GPIOStop();
00031
00032         struct Callback{
00033             virtual void MathsCallback(float X, float Y, float Z) = 0;
00034             virtual ~Callback(){};
00035         };
00036
00037         void RegisterCallback(Callback* cb){
00038             callback = cb;
00039         }
00040
00041         //Testing private stuff

```



```

00071     #ifdef UNIT_TEST
00072         bool GetRunning() const {
00073             return running.load();
00074         }
00075
00076         bool HasCallback() const {
00077             return callback != nullptr;
00078         }
00079
00080         Callback* GetCallback() const {
00081             return callback;
00082         }
00083     #endif
00084
00085     private:
00086     gpiod_chip* chip;
00087     gpiod_line* SensorLine;
00088     int InterruptPin;
00089     int Counter;
00090     bool Pause = true;
00091
00092     //Reference to IMU's driver
00093     icm20948::ICM20948_I2C& sensor;
00094
00095     std::atomic<bool> running{true};
00096     Callback* callback = nullptr;
00097
00098 };
00099
00100 struct MathsCallbackStruct : GPIOName::GPIOClass::Callback{
00101     IMUMathsName::IMUMaths& Maths;
00102
00103     MathsCallbackStruct(IMUMathsName::IMUMaths& maths) : Maths(maths) {}
00104
00105     virtual void MathsCallback(float X, float Y, float Z) override {
00106         Maths.SoundChecker(X, Y, Z);
00107     }
00108 };
00109
00110 }
00111
00112
00113
00114 #endif

```

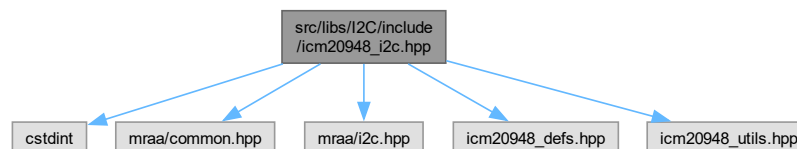
8.6 src/libs/I2C/include/icm20948_i2c.hpp File Reference

```

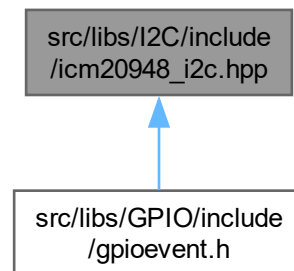
#include <stdint>
#include "mraa/common.hpp"
#include "mraa/i2c.hpp"
#include "icm20948_defs.hpp"
#include "icm20948_utils.hpp"

```

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
00025 bit);
00026         bool _read_bit(const uint8_t bank, const uint8_t reg, const uint8_t bit_pos, bool &bit);
00027         bool _read_block_bytes(const uint8_t bank, const uint8_t start_reg, uint8_t *bytes, const
00028 int length);
00029         bool _write_mag_byte(const uint8_t mag_reg, const uint8_t byte);
00030         bool _read_mag_byte(const uint8_t mag_reg, uint8_t &byte);
00031         bool _read_int_byte(const uint8_t bank, const uint8_t reg, uint8_t &byte);
00032
00033         bool _set_bank(uint8_t bank);
00034         bool _set_accel_sample_rate_div();
00035         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,
00066 icm20948::settings
00067             = icm20948::settings());
00068
00069         bool init();
00085
00086         bool reset();
00100
00101         bool wake();
00112
00113         bool set_settings();
00129
00130         bool read_accel_gyro();
00144
00145         bool read_magn();
00157
00158         bool enable_DRDY_INT();
00171
00172         bool check_DRDY_INT();
00182
00183     };
00184 }
00185
00186 #endif

```

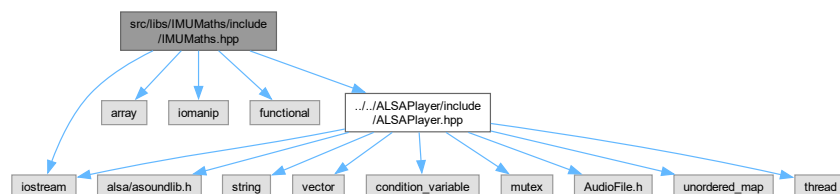
8.8 src/libs/IMUMaths/include/IMUMaths.hpp File Reference

```

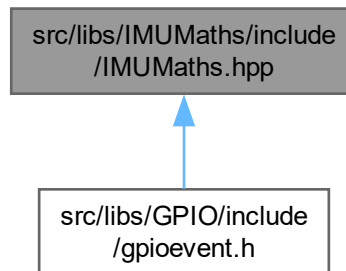
#include <iostream>
#include <array>
#include <iomanip>
#include <functional>
#include "../ALSAPlayer/include/ALSAPlayer.hpp"

```

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](#)
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.](#)

```

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
00028         void SoundChecker(float X, float Y, float Z);
00029
00030         // Pauses
00031         bool Pause = false;
00032
00033         // Counter variable
00034         int Counter = 0;
00035
00039         struct Callback{
00040             virtual void AudioTrigger(const std::string& FilePath) = 0;
00041             virtual ~Callback(){};
00042         };
  
```

```

00043
00044
00050     void RegisterCallback(Callback* cb){
00051         callback = cb;
00052         //std::cout << "[IMUMaths] Registered callback at address: " << callback << std::endl;
00053     }
00054
00055     //Access to private fo UNIT_TEST only
00056     #ifdef UNIT_TEST
00057
00058     bool HasCallback() const {
00059         return callback != nullptr;
00060     }
00061
00062     Callback* GetCallback() const {
00063         return callback;
00064     }
00065
00066     int LastFilePlayedTest() const{
00067         return LastFilePlayed;
00068     }
00069
00070     #endif
00071
00072
00073     private:
00074
00075     Callback* callback = nullptr;
00076     std::function<void(const std::string&)> PlayFileCallback;
00077
00078     // For debugging: Identifier of the last audio file played
00079
00080     int LastFilePlayed;
00081
00082 };
00083
00084
00085 struct AudioCallback : IMUMathsName::IMUMaths::Callback{
00086     AudioPlayerName::AudioPlayer& Audio;
00087
00088     AudioCallback(AudioPlayerName::AudioPlayer& audio) : Audio(audio) {}
00089
00090     virtual void AudioTrigger(const std::string& FilePath){
00091         std::thread([this,FilePath]{
00092             Audio.addSoundToMixer(FilePath);
00093
00094             }).detach();
00095     }
00096 };
00097 }
00098
00099
00100 #endif

```

