# SnAirBeats

1.0

Generated by Doxygen 1.13.2

# Chapter 1

# SnAirBeats

## 1.1 SnAIRbeats

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

## 1.2 Building

SnAIRBeats requires the following components to work:

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

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

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

## 1.3 Prerequisites

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

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

```
sudo apt update
```

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

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

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

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

## 1.4 Compliation from source

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

```
cmake .
make
```

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

```
./SnairBeats
```

## 1.5 Usage

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

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

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

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

## 1.6 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 the driver found at https://github.com/adamstark/AudioFile

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

### 1.6.2 GPIO

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

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

### 1.6.3 I2C

The I2C library is a driver written specifically for the ICM-20948 chip seen within the SEN 15335 IMU and is very heavily based off of driver written by NTKot found at https://github.com/NTkot/icm20948_i2c with the Raw-Data-Ready interrupt turned on and the magnetometer turned off.

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

### 1.6.4 IMUMaths

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

## 1.7 Unit tests

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

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

To run the tests from the root directory, use:

```
./run_tests
```

or to use CMake directly, run:

```
ctest
```

## 1.8 Sponsorship and funding

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

## 1.9 Media

- Instagram
- TikTok

## 1.10 Authors

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

## 1.11 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

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

# Chapter 5

# Namespace Documentation

## 5.1 AudioLib Namespace Reference

### Classes

- class AudioLib

## 5.2 AudioPlayerName Namespace Reference

### Classes

- class AudioPlayer

  *Handles audio file loading, conversion and playback.*

## 5.3 GPIOName Namespace Reference

### Classes

- class GPIOClass

### Typedefs

- typedef void(* GPIOCallback) (void *context, float, float, float)

### 5.3.1 Typedef Documentation

#### 5.3.1.1 GPIOCallback

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

## 5.4 icm20948 Namespace Reference

### Classes

- class ICM20948_I2C

## 5.5 IMUMathsName Namespace Reference

### Classes

- class IMUMaths

## 5.6 PlayAudioName Namespace Reference

### Classes

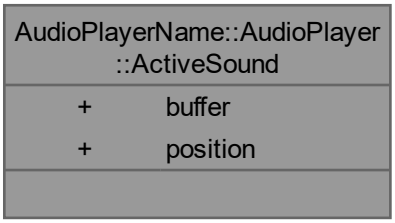
- class PlayAudio

# Chapter 6

# Class Documentation

## 6.1 AudioPlayerName::AudioPlayer::ActiveSound Struct Reference

Collaboration diagram for AudioPlayerName::AudioPlayer::ActiveSound:




### Public Attributes

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

### 6.1.1 Member Data Documentation

#### 6.1.1.1 buffer

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

#### 6.1.1.2 position

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

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

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

## 6.2 AudioLib::AudioLib Class Reference

`#include <AudioLib.hpp>`

Collaboration diagram for AudioLib::AudioLib:


### Public Member Functions

- AudioLib (const std::string &device="default")
- ∼AudioLib ()
- void PlaySound ()
- void PlayFile ()
- void PlayAudioTerminal ()

### Private Attributes

- snd_pcm_t ∗ pcmHandle = nullptr

### 6.2.1 Constructor & Destructor Documentation

#### 6.2.1.1 AudioLib()

```
AudioLib::AudioLib::AudioLib (
            const std::string & device = "default")
```

#### 6.2.1.2 ∼AudioLib()

```
AudioLib::AudioLib::~AudioLib ()
```

### 6.2.2 Member Function Documentation

#### 6.2.2.1 PlayAudioTerminal()

```
void AudioLib::AudioLib::PlayAudioTerminal ()
```

#### 6.2.2.2 PlayFile()

```
void AudioLib::AudioLib::PlayFile ()
```

#### 6.2.2.3 PlaySound()

```
void AudioLib::AudioLib::PlaySound ()
```

### 6.2.3 Member Data Documentation

#### 6.2.3.1 pcmHandle

```
snd_pcm_t* AudioLib::AudioLib::pcmHandle = nullptr  [private]
```

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

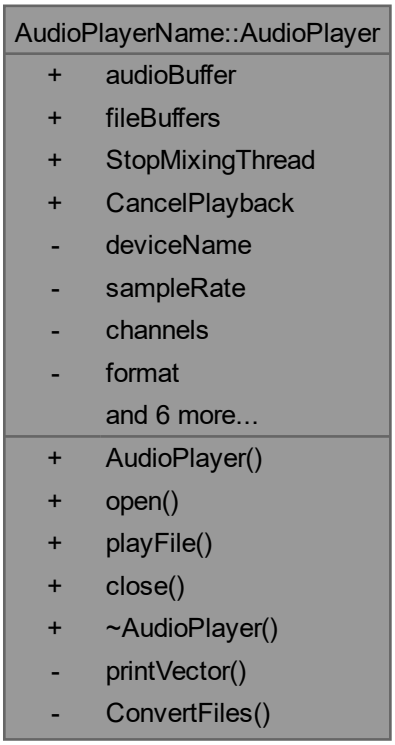
- src/libs/ALSAAudio/include/AudioLib.hpp

## 6.3 AudioPlayerName::AudioPlayer Class Reference

Handles audio file loading, conversion and playback.

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

Collaboration diagram for AudioPlayerName::AudioPlayer:




### Classes

- struct ActiveSound

### Public Member Functions

- AudioPlayer (const std::string &device="default", unsigned int rate=44100, unsigned int ch=2, snd_pcm_↩ format_t fmt=SND_PCM_FORMAT_S16_LE, snd_pcm_uframes_t frames=32, const std::vector< std::string > &filesToConvert={"src/libs/ALSAPlayer/include/CrashCymbal.wav", "src/libs/ALSAPlayer/include/High↩ Tom.wav", "src/libs/ALSAPlayer/include/SnareDrum.wav"})
- bool open ()

  *Open PCM device for playback.*
- bool playFile (const std::string &fileKey)

  *Play audio file using the PCM device.*
- void close ()

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

  *Destructor.*

### Public Attributes

- std::vector< int32_t > audioBuffer
- std::unordered_map< std::string, std::vector< int32_t > > fileBuffers
- bool StopMixingThread
- bool CancelPlayback = true

### Private Member Functions

- template<typename T>
  void printVector (const std::vector< T > &vec)
- void ConvertFiles (const std::vector< std::string > &filePaths)

  *Converts audio files to interleaved 32-bit int buffers for playback.*

### 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::condition_variable MixCV
- std::mutex MixCVMutex
- std::vector< ActiveSound > ActiveSounds
- std::mutex ActiveMutex

## 6.3.1 Detailed Description

Handles audio file loading, conversion and playback.

## 6.3.2 Constructor & Destructor Documentation

### 6.3.2.1 AudioPlayer()

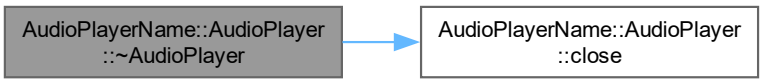
```
AudioPlayerName::AudioPlayer::AudioPlayer (
            const std::string & device = "default",
            unsigned int rate = 44100,
            unsigned int ch = 2,
            snd_pcm_format_t fmt = SND_PCM_FORMAT_S16_LE,
            snd_pcm_uframes_t frames = 32,
            const std::vector< std::string > & filesToConvert = {"src/libs/ALSAPlayer/include/CrashCymbal.wa
[inline]
```

#### 6.3.2.2 ~AudioPlayer()

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

Destructor.

Here is the call graph for this function:



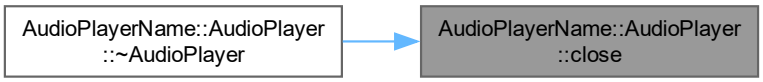

### 6.3.3 Member Function Documentation

#### 6.3.3.1 close()

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

Close PCM handle and free all associated resources.

Here is the caller graph for this function:




#### 6.3.3.2 ConvertFiles()

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

Converts audio files to interleaved 32-bit int buffers for playback.

**Parameters**

| | |
|---|---|
| *filePaths* | Path to audio files. |

#### 6.3.3.3 open()

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

Open PCM device for playback.

Opens the PCM device for playback and sets the hardware parameters. 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

#### 6.3.3.4 playFile()

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

Play audio file using the PCM device.

Plays an audio file using the PCM device. It includes the following steps:

- stop PCM playback and drop pending frames
- prepare PCM device for use
- check if the device is open and the audio buffer is available
- write the audio data to the PCM device in a loop until all frames are played
- close stream once all frames are played and put it in PREPARED state for next time

**Parameters**

| | |
|---|---|
| *fileKey* | The filename of the audio to be played. |

#### 6.3.3.5 printVector()

```
template<typename T>
void AudioPlayerName::AudioPlayer::printVector (
            const std::vector< T > & vec)  [inline], [private]
```

### 6.3.4 Member Data Documentation

#### 6.3.4.1 ActiveMutex

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

#### 6.3.4.2 ActiveSounds

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

#### 6.3.4.3 audioBuffer

```
std::vector<int32_t> AudioPlayerName::AudioPlayer::audioBuffer
```

#### 6.3.4.4 CancelPlayback

```
bool AudioPlayerName::AudioPlayer::CancelPlayback = true
```

#### 6.3.4.5 channels

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

#### 6.3.4.6 deviceName

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

#### 6.3.4.7 fileBuffers

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

#### 6.3.4.8 format

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

#### 6.3.4.9 framesPerPeriod

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

#### 6.3.4.10 handle

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

#### 6.3.4.11 MixCV

```
std::condition_variable AudioPlayerName::AudioPlayer::MixCV  [private]
```

#### 6.3.4.12 MixCVMutex

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

#### 6.3.4.13 sampleRate

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

#### 6.3.4.14 StopMixingThread

```
bool AudioPlayerName::AudioPlayer::StopMixingThread
```

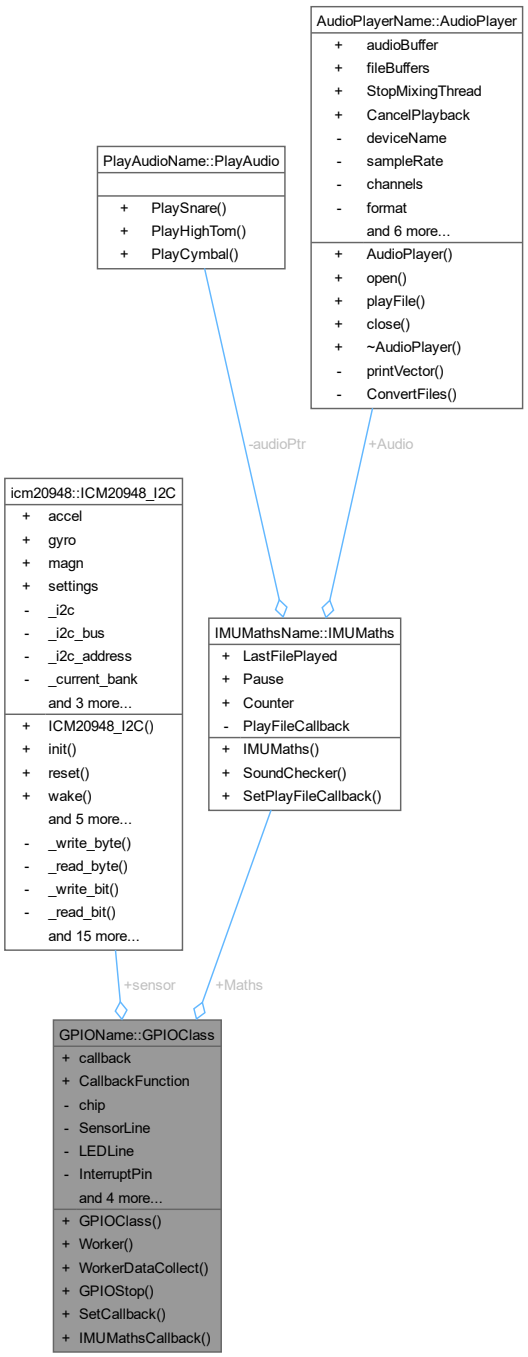
The documentation for this class was generated from the following file:

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

## 6.4 GPIOName::GPIOClass Class Reference

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

Collaboration diagram for GPIOName::GPIOClass:




## Public Member Functions

- GPIOClass (const char *chipName, int InterruptPin, icm20948::ICM20948_I2C &sensor, IMUMathsName::IMUMaths &Maths)
- 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 ()
- void SetCallback (GPIOCallback cb, void ∗context)

#### Static Public Member Functions

- static void IMUMathsCallback (void ∗context, float X, float Y, float Z)

#### Public Attributes

- icm20948::ICM20948_I2C & sensor
- IMUMathsName::IMUMaths & Maths
- GPIOCallback callback
- void ∗ CallbackFunction

#### Private Attributes

- gpiod_chip ∗ chip
- gpiod_line ∗ SensorLine
- gpiod_line ∗ LEDLine
- int InterruptPin
- int Counter
- bool Pause = true
- int delay = 224
- std::atomic< bool > running {true}

### 6.4.1 Constructor & Destructor Documentation

#### 6.4.1.1 GPIOClass()

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

### 6.4.2 Member Function Documentation

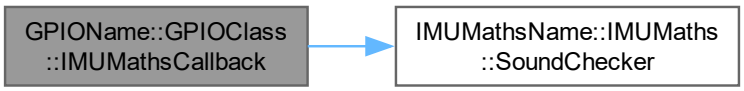
#### 6.4.2.1 GPIOStop()

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

#### 6.4.2.2 IMUMathsCallback()

```
static void GPIOName::GPIOClass::IMUMathsCallback (
            void * context,
            float X,
            float Y,
            float Z)  [inline], [static]
```

Here is the call graph for this function:




#### 6.4.2.3 SetCallback()

```
void GPIOName::GPIOClass::SetCallback (
            GPIOCallback cb,
            void * context)
```

#### 6.4.2.4 Worker()

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

Event driven worker reading data when HIGH seen on GPIO.

This function using blocking interrupts 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.

#### 6.4.2.5 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.

### 6.4.3 Member Data Documentation

#### 6.4.3.1 callback

```
GPIOCallback GPIOName::GPIOClass::callback
```

#### 6.4.3.2 CallbackFunction

```
void* GPIOName::GPIOClass::CallbackFunction
```

#### 6.4.3.3 chip

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

#### 6.4.3.4 Counter

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

#### 6.4.3.5 delay

```
int GPIOName::GPIOClass::delay = 224  [private]
```

#### 6.4.3.6 InterruptPin

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

#### 6.4.3.7 LEDLine

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

#### 6.4.3.8 Maths

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

#### 6.4.3.9 Pause

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

#### 6.4.3.10 running

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

#### 6.4.3.11 sensor

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

#### 6.4.3.12 SensorLine

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

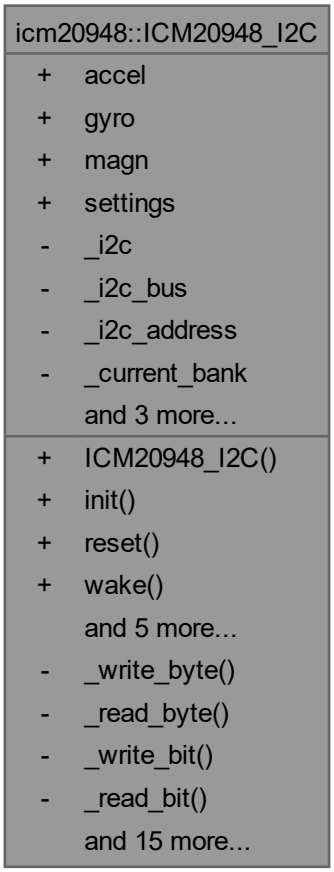
The documentation for this class was generated from the following file:

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

## 6.5 icm20948::ICM20948_I2C Class Reference

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

Collaboration diagram for icm20948::ICM20948_I2C:



### Public Member Functions

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

  *Initializes the ICM20948 sensor over I2C.*
- bool reset ()

  *Resets the ICM20948 sensor over I2C.*
- bool wake ()

  *Wakes up the ICM20948 sensor from sleep mode over I2C.*
- bool set_settings ()

  *Configures the ICM20948 sensor settings over I2C.*
- bool read_accel_gyro ()

  *Reads accelerometer and gyroscope data from the ICM20948 sensor over I2C.*
- bool read_magn ()

  *Reads magnetometer data from the ICM20948 sensor over I2C.*
- 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

### 6.5.1 Constructor & Destructor Documentation

#### 6.5.1.1 ICM20948_I2C()

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

### 6.5.2 Member Function Documentation

#### 6.5.2.1 _chip_i2c_master_reset()

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

#### 6.5.2.2 _magnetometer_configured()

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

#### 6.5.2.3 _magnetometer_enable()

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

#### 6.5.2.4 _magnetometer_init()

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

#### 6.5.2.5 _magnetometer_set_mode()

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

#### 6.5.2.6 _magnetometer_set_readout()

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

#### 6.5.2.7 _read_bit()

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

#### 6.5.2.8 _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]
```

#### 6.5.2.9 _read_byte()

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

#### 6.5.2.10 _read_int_byte()

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

#### 6.5.2.11 _read_mag_byte()

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

#### 6.5.2.12 _set_accel_range_dlpf()

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

#### 6.5.2.13 _set_accel_sample_rate_div()

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

#### 6.5.2.14 _set_bank()

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

#### 6.5.2.15 _set_gyro_range_dlpf()

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

#### 6.5.2.16 _set_gyro_sample_rate_div()

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

#### 6.5.2.17 _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]
```

#### 6.5.2.18 _write_byte()

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

#### 6.5.2.19 _write_mag_byte()

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

#### 6.5.2.20 check_DRDY_INT()

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

Checks if the Data Ready Interrupt is active.

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

**Returns**

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

#### 6.5.2.21 enable_DRDY_INT()

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

Enables the Data Ready Interrupt.

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

**Returns**

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

#### 6.5.2.22 init()

```
bool icm20948::ICM20948_I2C::init ()
```

Initializes the ICM20948 sensor over I2C.

This function performs the initialization sequence for the ICM20948 sensor. It includes the following steps:

- Selects Bank 0 of the ICM20948 registers.
- Reads the WHO_AM_I register to verify the sensor's identity.
- Resets the sensor to ensure it is in a known state.
- Wakes up the sensor from sleep mode.
- Configures the sensor settings (e.g., accelerometer, gyroscope settings).
- Attempts to initialize the magnetometer up to three times.

**Returns**

bool Returns true if the initialization sequence was successful, including successful magnetometer initialization. Returns false otherwise.

#### 6.5.2.23 read_accel_gyro()

```
bool icm20948::ICM20948_I2C::read_accel_gyro ()
```

Reads accelerometer and gyroscope data from the ICM20948 sensor over I2C.

This function reads a block of 12 bytes from the ICM20948 sensor, which includes the accelerometer and gyroscope data. It performs the following steps:

- Reads the accelerometer and gyroscope data from the sensor's registers.
- Reverses the byte order of the data for correct interpretation.
- Converts the raw accelerometer data to meters per second squared ($m/s^2$) using the configured scale factor.
- Converts the raw gyroscope data to radians per second (rad/s) using the configured scale factor.
- Stores the processed accelerometer data in the `accel` array and gyroscope data in the `gyro` array.

**Returns**

bool Returns true if the data was successfully read and processed. Returns false if the read operation fails.

#### 6.5.2.24 read_magn()

```
bool icm20948::ICM20948_I2C::read_magn ()
```

Reads magnetometer data from the ICM20948 sensor over I2C.

This function reads a block of 6 bytes from the ICM20948 sensor, which contains the magnetometer data. It performs the following steps:

- Reads the magnetometer data from the sensor's registers.
- Converts the raw magnetometer data to microteslas (μT) using the constant scale factor.
- Stores the processed magnetometer data in the `magn` array.

**Returns**

bool Returns true if the magnetometer data was successfully read and processed. Returns false if the read operation fails.

#### 6.5.2.25 reset()

```
bool icm20948::ICM20948_I2C::reset ()
```

Resets the ICM20948 sensor over I2C.

This function issues a reset command to the ICM20948 sensor and waits until the reset process is complete. It includes the following steps:

- Sets the reset bit in the PWR_MGMT_1 register to initiate a reset.
- Waits briefly (5 ms) to allow the reset to start.
- Polls the reset bit in the PWR_MGMT_1 register to check if the sensor is still resetting.
- Continues polling every 25 ms until the reset bit is cleared, indicating that the reset process is complete.
- Resets the internal bank tracking to Bank 0 after a successful reset.

**Returns**

bool Returns true if the reset process was successful. Returns false if any step in the reset process fails.

#### 6.5.2.26 set_settings()

```
bool icm20948::ICM20948_I2C::set_settings ()
```

Configures the ICM20948 sensor settings over I2C.

This function sets up various configuration parameters for the ICM20948 sensor, including:

- Accelerometer sample rate divider
- Accelerometer range and digital low-pass filter (DLPF) settings
- Gyroscope sample rate divider
- Gyroscope range and digital low-pass filter (DLPF) settings

Each configuration step is performed by calling the respective private methods. The overall success of the settings configuration is determined by the success of each individual step.

**Returns**

bool Returns true if all settings were successfully applied. Returns false if any configuration step fails.

#### 6.5.2.27 wake()

```
bool icm20948::ICM20948_I2C::wake ()
```

Wakes up the ICM20948 sensor from sleep mode over I2C.

This function clears the sleep bit in the PWR_MGMT_1 register to wake the ICM20948 sensor from sleep mode. It includes the following steps:

- Clears the sleep bit (bit 6) in the PWR_MGMT_1 register.
- Waits briefly (5 ms) to allow the sensor to stabilize after waking up.

**Returns**

bool Returns true if the wake-up process was successful. Returns false if the operation fails.

### 6.5.3 Member Data Documentation

#### 6.5.3.1 _accel_scale_factor

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

#### 6.5.3.2 _current_bank

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

#### 6.5.3.3 _gyro_scale_factor

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

#### 6.5.3.4 _i2c

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

#### 6.5.3.5 _i2c_address

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

#### 6.5.3.6 _i2c_bus

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

#### 6.5.3.7 _magn_scale_factor

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

#### 6.5.3.8 accel

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

#### 6.5.3.9 gyro

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

#### 6.5.3.10 magn

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

#### 6.5.3.11 settings

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

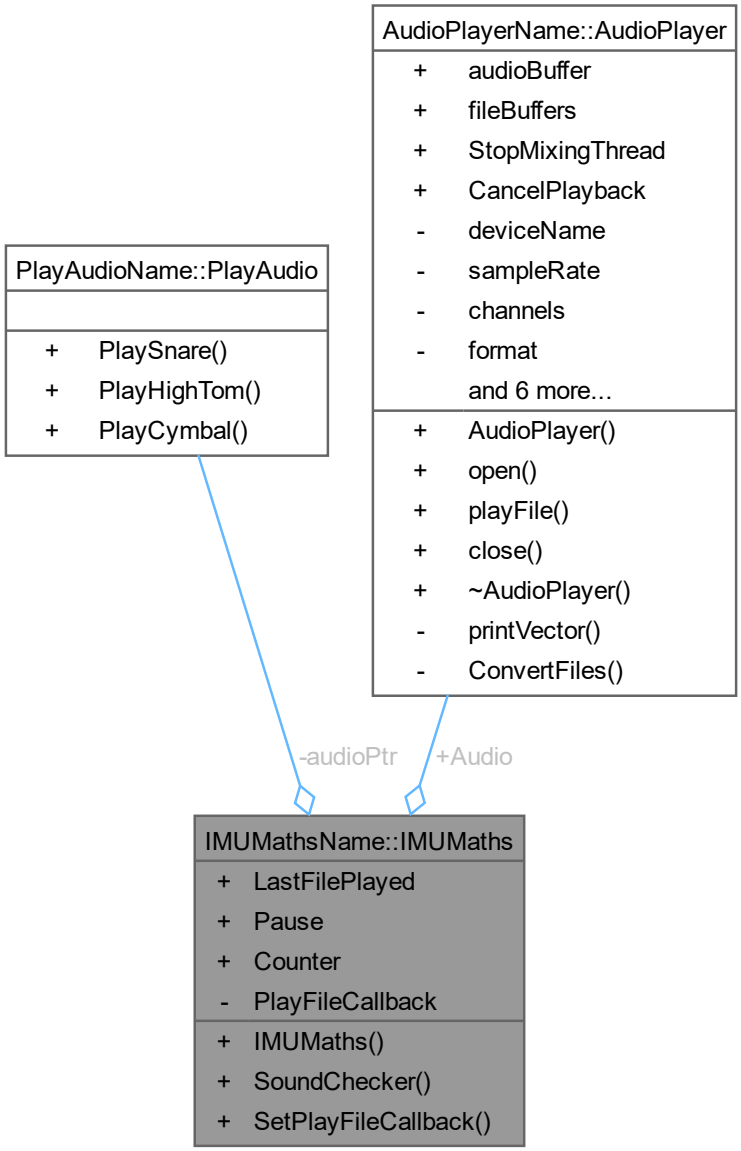
The documentation for this class was generated from the following file:

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

## 6.6 IMUMathsName::IMUMaths Class Reference

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

Collaboration diagram for IMUMathsName::IMUMaths:




### Public Member Functions

- IMUMaths (AudioPlayerName::AudioPlayer &Audio)

  *Constructs an object with access to the audio player.*
- 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.*

**Public Attributes**

- AudioPlayerName::AudioPlayer & Audio
- int LastFilePlayed
- bool Pause = false
- int Counter = 0

**Private Attributes**

- PlayAudioName::PlayAudio * audioPtr
- std::function< void(const std::string &)> PlayFileCallback

## 6.6.1 Constructor & Destructor Documentation

### 6.6.1.1 IMUMaths()

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

Constructs an object with access to the audio player.

**Parameters**

| | |
|---|---|
| *Audio* | used for playback |

## 6.6.2 Member Function Documentation

### 6.6.2.1 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**

| | |
|---|---|
| *cb* | |

### 6.6.2.2 SoundChecker()

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

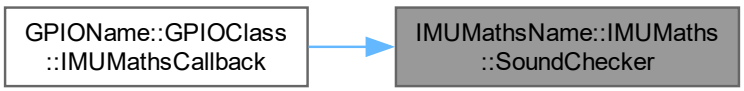
It measures each axis and sees if it falls within desired thresholds.

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

**Parameters**

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

Here is the caller graph for this function:




### 6.6.3 Member Data Documentation

#### 6.6.3.1 Audio

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

#### 6.6.3.2 audioPtr

```
PlayAudioName::PlayAudio* IMUMathsName::IMUMaths::audioPtr  [private]
```

#### 6.6.3.3 Counter

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

#### 6.6.3.4 LastFilePlayed

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

#### 6.6.3.5 Pause

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

#### 6.6.3.6 PlayFileCallback

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

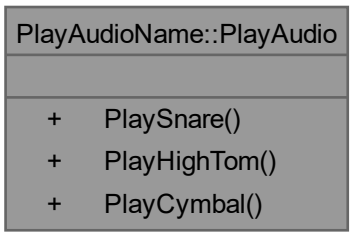
The documentation for this class was generated from the following file:

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

## 6.7 PlayAudioName::PlayAudio Class Reference

`#include <PlayAudio.hpp>`

Collaboration diagram for PlayAudioName::PlayAudio:




### Static Public Member Functions

- static void PlaySnare ()
- static void PlayHighTom ()
- static void PlayCymbal ()

### 6.7.1 Member Function Documentation

#### 6.7.1.1 PlayCymbal()

`static void PlayAudioName::PlayAudio::PlayCymbal () [static]`

#### 6.7.1.2 PlayHighTom()

`static void PlayAudioName::PlayAudio::PlayHighTom () [static]`

#### 6.7.1.3 PlaySnare()

`static void PlayAudioName::PlayAudio::PlaySnare () [static]`

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

- src/libs/PlayAudio/include/PlayAudio.hpp

# Chapter 7

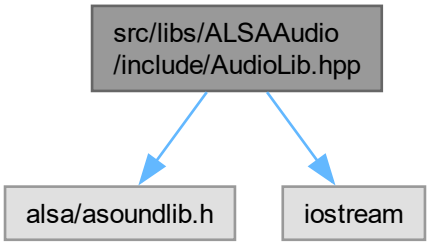
# File Documentation

## 7.1 README.md File Reference

## 7.2 src/libs/ALSAAudio/include/AudioLib.hpp File Reference

```
#include <alsa/asoundlib.h>
#include <iostream>
```

Include dependency graph for AudioLib.hpp:




### Classes

- class AudioLib::AudioLib

### Namespaces

- namespace AudioLib

## 7.3 AudioLib.hpp

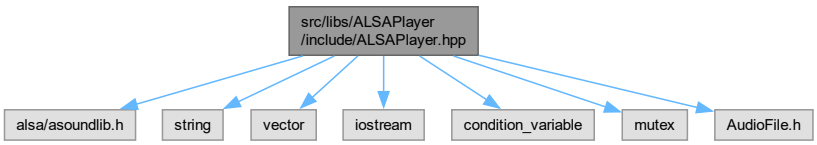
Go to the documentation of this file.

```
00001 #ifndef AUDIOLIB_H
00002 #define AUDIOLIB_H
00003
00004 #include <alsa/asoundlib.h>
00005 #include <iostream>
00006
00007 namespace AudioLib {
00008     class AudioLib {
00009     private:
00010         snd_pcm_t *pcmHandle= nullptr;
00011
00012     public:
00013     AudioLib(const std::string &device = "default");
00014     ~AudioLib();
00015
00016     void PlaySound();
00017     void PlayFile();
00018     void PlayAudioTerminal();
00019     };
00020 }
00021
00022
00023
00024
00025
00026 #endif
```

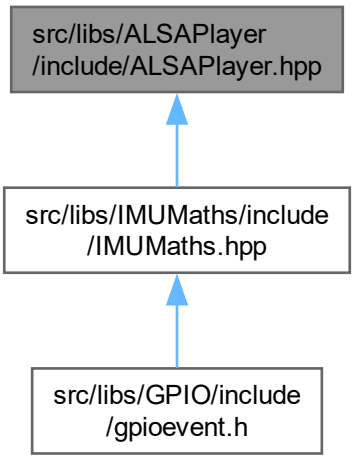
## 7.4 src/libs/ALSAPlayer/include/ALSAPlayer.hpp File Reference

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

Include dependency graph for ALSAPlayer.hpp:



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




### Classes

- class AudioPlayerName::AudioPlayer

  *Handles audio file loading, conversion and playback.*
- struct AudioPlayerName::AudioPlayer::ActiveSound

### Namespaces

- namespace AudioPlayerName

## 7.5 ALSAPlayer.hpp

Go to the documentation of this file.

```
00001 #ifndef ALSAPLAYER_H
00002 #define ALSAPLAYER_H
00003
00004 #include <alsa/asoundlib.h>
00005 #include <string>
00006 #include <vector>
00007 #include <iostream>
00008 #include <condition_variable>
00009 #include <mutex>
00010 #include "AudioFile.h"
00011
00012
00013
00014 namespace AudioPlayerName{
00015
00019     class AudioPlayer{
00020
00021         public:
00022         std::vector<int32_t> audioBuffer;
00023         std::unordered_map<std::string, std::vector<int32_t» fileBuffers;
00024
00025         bool StopMixingThread;
00026         bool CancelPlayback = true;
```

```
00027
00028
00029         AudioPlayer(const std::string& device="default",
00030             unsigned int rate = 44100,
00031             unsigned int ch = 2,
00032             snd_pcm_format_t fmt = SND_PCM_FORMAT_S16_LE,
00033             snd_pcm_uframes_t frames = 32,
00034             const std::vector<std::string>& filesToConvert =
      {"src/libs/ALSAPlayer/include/CrashCymbal.wav",
00035
      "src/libs/ALSAPlayer/include/HighTom.wav",
00036
      "src/libs/ALSAPlayer/include/SnareDrum.wav"})
00037         : deviceName(device), sampleRate(rate), channels(ch),
00038         format(fmt), framesPerPeriod(frames), handle(nullptr)
00039         {
00040             //Convert files to audio buffers
00041             if (!filesToConvert.empty()){
00042                 ConvertFiles(filesToConvert);
00043             }
00044         }
00045
00057         bool open(){
00058             int rc = snd_pcm_open(&handle, deviceName.c_str(), SND_PCM_STREAM_PLAYBACK,0);
00059             if (rc < 0){
00060                 std::cerr « "Unable to open PCM devices: " « snd_strerror(rc) « std::endl;
00061                 return false;
00062             }
00063
00064             snd_pcm_hw_params_t* params;
00065             snd_pcm_hw_params_alloca(&params);
00066             snd_pcm_hw_params_any(handle, params);
00067             snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
00068             snd_pcm_hw_params_set_format(handle, params, format);
00069             snd_pcm_hw_params_set_channels(handle, params, channels);
00070
00071             unsigned int rate_near = sampleRate;
00072             snd_pcm_hw_params_set_rate_near(handle, params, &rate_near,0);
00073             snd_pcm_hw_params_set_period_size_near(handle, params, &framesPerPeriod, 0);
00074
00075             rc = snd_pcm_hw_params(handle, params);
00076             if (rc <0) {
00077                 std::cerr « "Unable to set HW parameters: " « snd_strerror(rc) « std::endl;
00078                 return false;
00079             }
00080
00081             snd_pcm_hw_params_get_period_size(params, &framesPerPeriod,0);
00082             return true;
00083         }
00084
00097         bool playFile(const std::string& fileKey) {
00098
00099             CancelPlayback = true;
00100
00101             if(handle) {
00102                 snd_pcm_drop(handle);
00103                 snd_pcm_prepare(handle);
00104             }
00105
00106             CancelPlayback = false;
00107
00108             if (!handle) {
00109                 std::cerr « "Device not open. Call open() first.\n";
00110                 return false;
00111             }
00112             if (fileBuffers.find(fileKey) == fileBuffers.end()) {
00113                 std::cerr « "Audio buffer not found for file: " « fileKey « "\n";
00114                 return false;
00115             }
00116
00117             const std::vector<int32_t>& buffer = fileBuffers[fileKey];
00118             size_t totalFrames = buffer.size() / channels;
00119             size_t offset = 0;
00120             int rc = 0;
00121
00122             if (CancelPlayback){
00123                 std::cerr « "[DEBUG] Playback cancelled." « std::endl;
00124                 return false;
00125             }
00126
00127
00128             while (offset < totalFrames) {
00129                 snd_pcm_uframes_t framesToWrite = framesPerPeriod;
00130                 if (offset + framesPerPeriod > totalFrames)
00131                     framesToWrite = totalFrames - offset;
00132                 rc = snd_pcm_writei(handle, buffer.data() + offset * channels, framesToWrite);
00133                 if (rc == -EPIPE) {
```

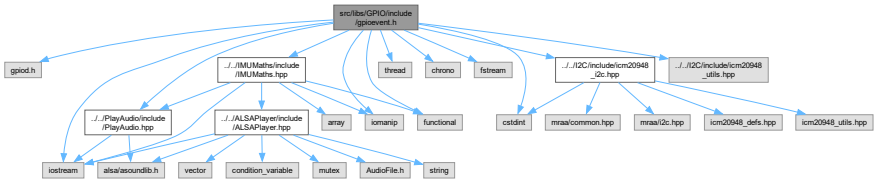
```
00134                     std::cerr « "Underrun occurred\n";
00135                     snd_pcm_prepare(handle);
00136                 } else if (rc < 0) {
00137                     std::cerr « "Error from writei: " « snd_strerror(rc) « "\n";
00138                     return false;
00139                 } else if (static_cast<snd_pcm_uframes_t>(rc) != framesToWrite) {
00140                     std::cerr « "Short write, wrote " « rc « " frames\n";
00141                 } else {
00142                     offset += rc;
00143                 }
00144             }
00145             snd_pcm_drain(handle);
00146             snd_pcm_prepare(handle);
00147             return true;
00148         }
00149 
00153         void close() {
00154             if (handle) {
00155                 snd_pcm_close(handle);
00156                 handle = nullptr;
00157             }
00158         }
00159 
00163         ~AudioPlayer() {
00164             close();
00165         }
00166 
00167 
00168         private:
00169         std::string deviceName;
00170         unsigned int sampleRate;
00171         unsigned int channels;
00172         snd_pcm_format_t format;
00173         snd_pcm_uframes_t framesPerPeriod;
00174         snd_pcm_t* handle;
00175 
00176         std::condition_variable MixCV;
00177         std::mutex MixCVMutex;
00178 
00179         struct ActiveSound {
00180             std::vector<int32_t>* buffer;
00181             size_t position;
00182         };
00183 
00184         std::vector<ActiveSound> ActiveSounds;
00185         std::mutex ActiveMutex;
00186 
00187 
00188         template<typename T>
00189         void printVector(const std::vector<T>& vec) {
00190         for (const auto& el : vec) {
00191             std::cout « el « " ";
00192         }
00193         std::cout « std::endl;
00194         }
00195 
00200         void ConvertFiles(const std::vector<std::string>& filePaths) {
00201             std::vector<int32_t> result;
00202 
00203             for (const auto& path : filePaths) {
00204                 AudioFile<int32_t> file;
00205                 if (!file.load(path)) {
00206                     std::cerr « "Error loading file: " « path « std::endl;
00207                     continue;
00208                 }
00209 
00210                 int fileChannels = file.getNumChannels();
00211                 int ChannelSamples = file.getNumSamplesPerChannel();
00212 
00213                 std::vector<int32_t> interleaved;
00214                 interleaved.reserve(ChannelSamples * fileChannels);
00215                 for (int i=0; i < ChannelSamples; ++i){
00216                     for (int ch = 0; ch < fileChannels; ++ch){
00217                         interleaved.push_back(file.samples[ch][i]);
00218 
00219                     }
00220 
00221 
00222                 }
00223                 //printVector(interleaved);
00224                 fileBuffers[path] = std::move(interleaved);
00225 
00226             }
00227         }
00228 
00229 
00230     };
```

```
00231 }
00232
00233
00234
00235 #endif
```

## 7.6 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 "../../I2C/include/icm20948_i2c.hpp"
#include "../../I2C/include/icm20948_utils.hpp"
#include "../../IMUMaths/include/IMUMaths.hpp"
#include "../../PlayAudio/include/PlayAudio.hpp"
```

Include dependency graph for gpioevent.h:




### Classes

- class GPIOName::GPIOClass

### Namespaces

- namespace GPIOName

### Typedefs

- typedef void(∗ GPIOName::GPIOCallback) (void ∗context, float, float, float)

## 7.7 gpioevent.h

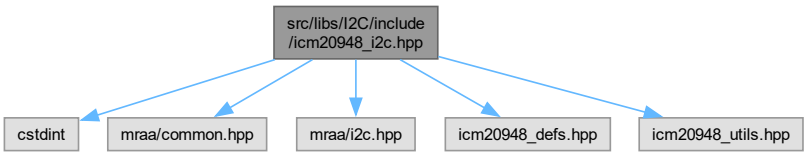
Go to the documentation of this file.

```
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
00014 #include "../../I2C/include/icm20948_i2c.hpp"
00015 #include "../../I2C/include/icm20948_utils.hpp"
00016
00017 #include "../../IMUMaths/include/IMUMaths.hpp"
00018
00019 #include "../../PlayAudio/include/PlayAudio.hpp"
00020
00021 namespace GPIOName {
00022
00023     typedef void (*GPIOCallback)(void* context, float, float, float);
00024     class GPIOClass {
00025     private:
00026         gpiod_chip* chip;
00027         gpiod_line* SensorLine;
00028         gpiod_line* LEDLine;
00029         int InterruptPin;
00030         int Counter;
00031         bool Pause = true;
00032         int delay = 224;
00033         std::atomic<bool> running{true};
00034
00035     public:
00036         icm20948::ICM20948_I2C& sensor;
00037         IMUMathsName::IMUMaths& Maths;
00038
00039         GPIOCallback callback;
00040         void* CallbackFunction;
00041
00042         //Constructor
00043         GPIOClass(const char* chipName, int InterruptPin,
00044              icm20948::ICM20948_I2C& sensor, IMUMathsName::IMUMaths& Maths);
00045
00054         void Worker();
00055
00066         void WorkerDataCollect();
00067         void GPIOStop();
00068         void SetCallback(GPIOCallback cb, void* context);
00069         static void IMUMathsCallback(void* context, float X, float Y, float Z){
00070             IMUMathsName::IMUMaths* maths = static_cast<IMUMathsName::IMUMaths*>(context);
00071             maths->SoundChecker(X,Y,Z);
00072         }
00073
00074     };
00075 }
00076
00077 #endif
```

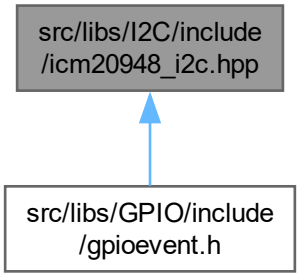
## 7.8 src/libs/I2C/include/icm20948_i2c.hpp File Reference

```
#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

## 7.9 icm20948_i2c.hpp

Go to the documentation of this file.

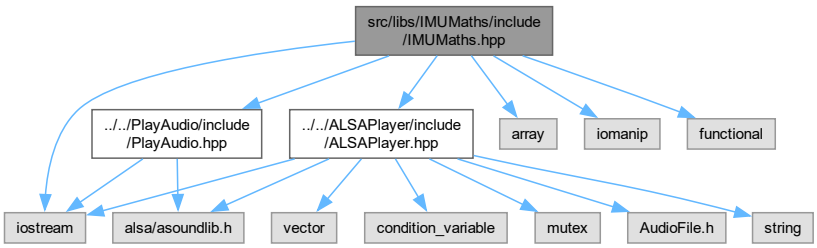
```
00001 #ifndef ICM20948_I2C_HPP
00002 #define ICM20948_I2C_HPP
00003
00004 #include <cstdint>
00005
00006 #include "mraa/common.hpp"
00007 #include "mraa/i2c.hpp"
00008
00009 #include "icm20948_defs.hpp"
00010 #include "icm20948_utils.hpp"
00011
00012 namespace icm20948
00013 {
00014     class ICM20948_I2C
```

```
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 
00056             // Constructor
00057             ICM20948_I2C(unsigned i2c_bus, unsigned i2c_address = ICM20948_I2C_ADDR,
      icm20948::settings
00058                     = icm20948::settings());
00059 
00075             bool init();
00076 
00090             bool reset();
00091 
00102             bool wake();
00103 
00119             bool set_settings();
00120 
00134             bool read_accel_gyro();
00135 
00147             bool read_magn();
00148 
00159             bool enable_DRDY_INT();
00160 
00170             bool check_DRDY_INT();
00171     };
00172 }
00173 
00174 #endif
```

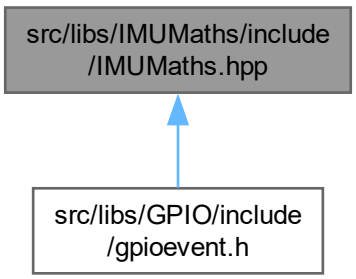
## 7.10 src/libs/IMUMaths/include/IMUMaths.hpp File Reference

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

### Namespaces

- namespace IMUMathsName

## 7.11 IMUMaths.hpp

Go to the documentation of this file.

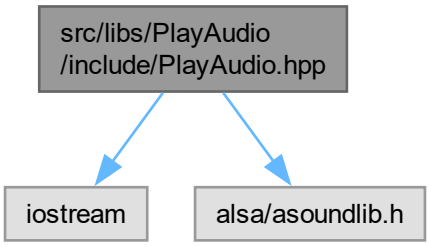
```
00001 #ifndef IMUMATHS_H
00002 #define IMUMATHS_H
00003
00004
00005 #include <iostream>
00006 #include <array>
00007 #include <iomanip>
00008 #include <functional>
00009
00010 #include "../../PlayAudio/include/PlayAudio.hpp"
```

```
00011
00012 #include "../../ALSAPlayer/include/ALSAPlayer.hpp"
00013
00014
00015 namespace IMUMathsName {
00016     class IMUMaths{
00017         private:
00018
00019         PlayAudioName::PlayAudio* audioPtr;
00020         std::function<void(const std::string&)> PlayFileCallback;
00021
00022         public:
00023         AudioPlayerName::AudioPlayer &Audio;
00024
00030         IMUMaths(AudioPlayerName::AudioPlayer &Audio);
00031
00032         // For debugging: Identifier of the last audio file played
00033         int LastFilePlayed;
00034
00045         void SoundChecker(float X, float Y, float Z);
00046
00054         void SetPlayFileCallback(const std::function<void(const std::string&)>& cb);
00055
00056         // Pauses
00057         bool Pause = false;
00058
00059         // Counter variable
00060         int Counter = 0;
00061
00062
00063     };
00064 }
00065
00066
00067 #endif
```

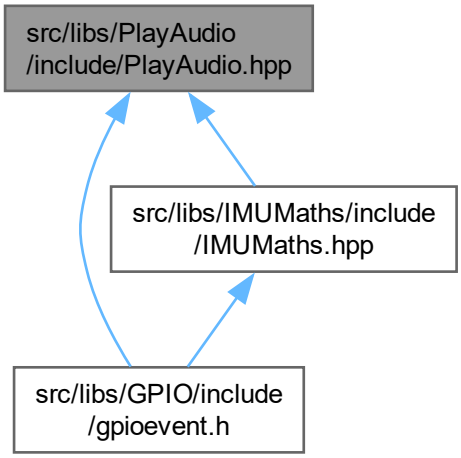
## 7.12 src/libs/PlayAudio/include/PlayAudio.hpp File Reference

```
#include <iostream>
#include <alsa/asoundlib.h>
```

Include dependency graph for PlayAudio.hpp:



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




### Classes

- class PlayAudioName::PlayAudio

### Namespaces

- namespace PlayAudioName

## 7.13 PlayAudio.hpp

Go to the documentation of this file.

```
00001 #ifndef PLAYAUDIO_H
00002 #define PLAYAUDIO_H
00003
00004 #include <iostream>
00005 #include <alsa/asoundlib.h>
00006
00007 namespace PlayAudioName{
00008     class PlayAudio{
00009         public:
00010
00011         static void PlaySnare();
00012         static void PlayHighTom();
00013         static void PlayCymbal();
00014     };
00015 }
00016
00017
00018
00019
00020
00021 #endif
```