# SnAirBeats

1.0

Generated by Doxygen 1.13.2

# Chapter 1

# SnAirBeats

## 1.1 SnAIRbeats

SnAirBeats is a next generation methods to practice the drum kit, while reducing noise and space 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 -ylibyaml-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.

Audio devices are opened using the Open function which once finished can be used to player 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 and contributions

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

## 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

# 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

## 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

- struct accel_settings

  *Configuration settings for an accelerometer sensor.*
- struct gyro_settings

  *Configuration settings for a gyroscope sensor.*
- class ICM20948_I2C
- struct magn_settings

  *Configuration settings for a magnetometer sensor.*
- struct settings

  *Aggregated configuration settings for sensor modules.*

### Typedefs

- typedef struct icm20948::accel_settings accel_settings
- typedef struct icm20948::gyro_settings gyro_settings
- typedef struct icm20948::magn_settings magn_settings
- typedef struct icm20948::settings settings

### Enumerations

- enum accel_scale { ACCEL_2G = 0 , ACCEL_4G , ACCEL_8G , ACCEL_16G }

  *Represents the accelerometer full-scale range settings.*
- enum accel_dlpf_config {
  ACCEL_DLPF_246HZ = 0 , ACCEL_DLPF_246HZ_2 , ACCEL_DLPF_111_4HZ , ACCEL_DLPF_50_4HZ ,
  ACCEL_DLPF_23_9HZ , ACCEL_DLPF_11_5HZ , ACCEL_DLPF_5_7HZ , ACCEL_DLPF_473HZ }

  *Represents the accelerometer Digital Low Pass Filter (DLPF) configuration settings.*
- enum gyro_scale { GYRO_250DPS = 0 , GYRO_500DPS , GYRO_1000DPS , GYRO_2000DPS }

  *Represents the gyroscope full-scale range settings.*
- enum gyro_dlpf_config {
  GYRO_DLPF_196_6HZ = 0 , GYRO_DLPF_151_8HZ , GYRO_DLPF_119_5HZ , GYRO_DLPF_51_2HZ ,
  GYRO_DLPF_23_9HZ , GYRO_DLPF_11_6HZ , GYRO_DLPF_5_7HZ , GYRO_DLPF_361_4HZ }

  *Represents the gyroscope Digital Low Pass Filter (DLPF) configuration settings.*
- enum magn_mode {
  MAGN_SHUTDOWN = 0 , MAGN_SINGLE = 1 , MAGN_10HZ = 2 , MAGN_20HZ = 4 ,
  MAGN_50HZ = 6 , MAGN_100HZ = 8 , MAGN_SELF_TEST = 16 }

  *Represents the magnetometer operating modes.*

### Functions

- float accel_scale_factor (accel_scale scale)

  *Calculates the scale factor for accelerometer measurements.*
- std::string accel_scale_to_str (accel_scale scale)

  *Converts an accelerometer scale value to its corresponding string representation.*
- std::string accel_dlpf_config_to_str (accel_dlpf_config config)

  *Converts an accelerometer Digital Low-Pass Filter (DLPF) configuration value to its corresponding string representation.*
- float gyro_scale_factor (gyro_scale scale)

*Calculates the scale factor for gyroscope measurements.*

- std::string gyro_scale_to_str (gyro_scale scale)

  *Converts a gyroscope scale value to its corresponding string representation.*
- std::string gyro_dlpf_config_to_str (gyro_dlpf_config config)

  *Converts a gyroscope Digital Low-Pass Filter (DLPF) configuration value to its corresponding string representation.*
- std::string magn_mode_to_str (magn_mode mode)

  *Converts a magnetometer operation mode value to its corresponding string representation.*

## 5.4.1 Typedef Documentation

### 5.4.1.1 accel_settings

```
typedef struct icm20948::accel_settings icm20948::accel_settings
```

### 5.4.1.2 gyro_settings

```
typedef struct icm20948::gyro_settings icm20948::gyro_settings
```

### 5.4.1.3 magn_settings

```
typedef struct icm20948::magn_settings icm20948::magn_settings
```

### 5.4.1.4 settings

```
typedef struct icm20948::settings icm20948::settings
```

## 5.4.2 Enumeration Type Documentation

### 5.4.2.1 accel_dlpf_config

```
enum icm20948::accel_dlpf_config
```

Represents the accelerometer Digital Low Pass Filter (DLPF) configuration settings.

This enumeration defines the different Digital Low Pass Filter (DLPF) configurations that can be selected for the accelerometer. The DLPF is used to filter out high-frequency noise from the sensor's measurements, with different cutoff frequencies available depending on the application requirements.

The possible values are:

- ACCEL_DLPF_246HZ: 246 Hz DLPF cutoff frequency
- ACCEL_DLPF_246HZ_2: Alternate 246 Hz DLPF cutoff frequency
- ACCEL_DLPF_111_4HZ: 111.4 Hz DLPF cutoff frequency
- ACCEL_DLPF_50_4HZ: 50.4 Hz DLPF cutoff frequency
- ACCEL_DLPF_23_9HZ: 23.9 Hz DLPF cutoff frequency
- ACCEL_DLPF_11_5HZ: 11.5 Hz DLPF cutoff frequency
- ACCEL_DLPF_5_7HZ: 5.7 Hz DLPF cutoff frequency
- ACCEL_DLPF_473HZ: 473 Hz DLPF cutoff frequency

**Enumerator**

| | |
|---|---|
| ACCEL_DLPF_246HZ | |
| ACCEL_DLPF_246HZ_2 | |
| ACCEL_DLPF_111_4HZ | |
| ACCEL_DLPF_50_4HZ | |
| ACCEL_DLPF_23_9HZ | |
| ACCEL_DLPF_11_5HZ | |
| ACCEL_DLPF_5_7HZ | |
| ACCEL_DLPF_473HZ | |

#### 5.4.2.2 accel_scale

`enum icm20948::accel_scale`

Represents the accelerometer full-scale range settings.

This enumeration defines the different full-scale ranges that can be selected for the accelerometer. The full-scale range determines the maximum measurable acceleration by the sensor in terms of gravitational force (g).

The possible values are:

- ACCEL_2G: +/- 2g range
- ACCEL_4G: +/- 4g range
- ACCEL_8G: +/- 8g range
- ACCEL_16G: +/- 16g range

**Enumerator**

| | |
|---|---|
| ACCEL_2G | |
| ACCEL_4G | |
| ACCEL_8G | |
| ACCEL_16G | |

#### 5.4.2.3 gyro_dlpf_config

`enum icm20948::gyro_dlpf_config`

Represents the gyroscope Digital Low Pass Filter (DLPF) configuration settings.

This enumeration defines the different Digital Low Pass Filter (DLPF) configurations that can be selected for the gyroscope. The DLPF is used to filter out high-frequency noise from the sensor's measurements, with various cutoff frequencies available
depending on the desired noise reduction and signal fidelity.

The possible values are:

- GYRO_DLPF_196_6HZ: 196.6 Hz DLPF cutoff frequency
- GYRO_DLPF_151_8HZ: 151.8 Hz DLPF cutoff frequency
- GYRO_DLPF_119_5HZ: 119.5 Hz DLPF cutoff frequency
- GYRO_DLPF_51_2HZ: 51.2 Hz DLPF cutoff frequency
- GYRO_DLPF_23_9HZ: 23.9 Hz DLPF cutoff frequency
- GYRO_DLPF_11_6HZ: 11.6 Hz DLPF cutoff frequency
- GYRO_DLPF_5_7HZ: 5.7 Hz DLPF cutoff frequency
- GYRO_DLPF_361_4HZ: 361.4 Hz DLPF cutoff frequency

**Enumerator**

| | |
|---|---|
| GYRO_DLPF_196_6HZ | |
| GYRO_DLPF_151_8HZ | |
| GYRO_DLPF_119_5HZ | |
| GYRO_DLPF_51_2HZ | |
| GYRO_DLPF_23_9HZ | |
| GYRO_DLPF_11_6HZ | |
| GYRO_DLPF_5_7HZ | |
| GYRO_DLPF_361_4HZ | |

#### 5.4.2.4 gyro_scale

```
enum icm20948::gyro_scale
```

Represents the gyroscope full-scale range settings.

This enumeration defines the different full-scale ranges that can be selected for the gyroscope. The full-scale range determines the maximum measurable angular velocity by the sensor in degrees per second (DPS).

The possible values are:

- GYRO_250DPS: 250 degrees per second full-scale range
- GYRO_500DPS: 500 degrees per second full-scale range
- GYRO_1000DPS: 1000 degrees per second full-scale range
- GYRO_2000DPS: 2000 degrees per second full-scale range

**Enumerator**

| | |
|---|---|
| GYRO_250DPS | |
| GYRO_500DPS | |
| GYRO_1000DPS | |
| GYRO_2000DPS | |

#### 5.4.2.5 magn_mode

```
enum icm20948::magn_mode
```

Represents the magnetometer operating modes.

This enumeration defines the different operating modes for the magnetometer, which control the sensor's measurement mode and update rate. Each mode
specifies a different operational state or measurement frequency.

The possible values are:

- MAGN_SHUTDOWN: Shutdown mode, which powers down the sensor to save energy.
- MAGN_SINGLE: Single measurement mode, where the sensor takes a single reading and then enters a low-power state.
- MAGN_10HZ: Measurement mode with a 10 Hz inner sample rate, providing updates at 10 times per second.
- MAGN_20HZ: Measurement mode with a 20 Hz inner sample rate, providing updates at 20 times per second.
- MAGN_50HZ: Measurement mode with a 50 Hz inner sample rate, providing updates at 50 times per second.
- MAGN_100HZ: Measurement mode with a 100 Hz inner sample rate, providing updates at 100 times per second.
- MAGN_SELF_TEST: Self-test mode, used for performing internal sensor diagnostics and calibration.

**Enumerator**

| | |
|---|---|
| MAGN_SHUTDOWN | |
| MAGN_SINGLE | |
| MAGN_10HZ | |
| MAGN_20HZ | |
| MAGN_50HZ | |
| MAGN_100HZ | |
| MAGN_SELF_TEST | |

### 5.4.3 Function Documentation

#### 5.4.3.1 accel_dlpf_config_to_str()

```
std::string icm20948::accel_dlpf_config_to_str (
            accel_dlpf_config config)
```

Converts an accelerometer Digital Low-Pass Filter (DLPF) configuration value to its corresponding string representation.

This function maps an `accel_dlpf_config` enumeration value to a human-readable string that represents the accelerometer's DLPF cutoff frequency.

**Parameters**

| | |
|---|---|
| *config* | The Digital Low-Pass Filter configuration for the accelerometer, defined by the accel_dlpf_config enumeration. Valid values include ACCEL_DLPF_246HZ, ACCEL_DLPF_246HZ_2, ACCEL_DLPF_111_4HZ, ACCEL_DLPF_50_4HZ, ACCEL_DLPF_23_9HZ, ACCEL_DLPF_11_5HZ, ACCEL_DLPF_5_7HZ, and ACCEL_DLPF_473HZ. |

**Returns**

A string representing the accelerometer DLPF configuration. The possible return values are "246Hz", "111.↩ 4Hz", "50.4Hz", "23.9Hz", "11.5Hz", "5.7Hz", and "473Hz". If an invalid accel_dlpf_config value is provided, the function returns "<invalid accelerometer DLPF config>".

### 5.4.3.2 accel_scale_factor()

```
float icm20948::accel_scale_factor (
            accel_scale scale)
```

Calculates the scale factor for accelerometer measurements.

This function determines the scale factor for the accelerometer based on the selected full-scale range. The scale factor is used to convert raw sensor readings into meaningful acceleration values in units of g (gravitational force).

**Parameters**

| | |
|---|---|
| *scale* | The full-scale range setting for the accelerometer, defined by the accel_scale enumeration. Valid values include ACCEL_2G, ACCEL_4G, ACCEL_8G, and ACCEL_16G. |

**Returns**

The scale factor (float) corresponding to the specified accelerometer full-scale range. The scale factor is expressed as the reciprocal (+/-) of the full-scale measurement range in terms of g.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If an invalid accel_scale value is provided. |

### 5.4.3.3 accel_scale_to_str()

```
std::string icm20948::accel_scale_to_str (
            accel_scale scale)
```

Converts an accelerometer scale value to its corresponding string representation.

This function maps an accel_scale enumeration value to a human-readable string that represents the accelerometer's full-scale range setting.

**Parameters**

| | |
|---|---|
| *scale* | The full-scale range setting for the accelerometer, defined by the accel_scale enumeration. Valid values include ACCEL_2G, ACCEL_4G, ACCEL_8G, and ACCEL_16G. |

**Returns**

A string representing the accelerometer scale. The possible return values are "2G", "4G", "8G", and "16G". If an invalid accel_scale value is provided, the function returns "<invalid accelerometer scale>".

#### 5.4.3.4 gyro_dlpf_config_to_str()

```
std::string icm20948::gyro_dlpf_config_to_str (
            gyro_dlpf_config config)
```

Converts a gyroscope Digital Low-Pass Filter (DLPF) configuration value to its corresponding string representation.

This function maps a gyro_dlpf_config enumeration value to a human-readable string that represents the gyroscope's DLPF cutoff frequency.

**Parameters**

| | |
|---|---|
| *config* | The Digital Low-Pass Filter configuration for the gyroscope, defined by the gyro_dlpf_config enumeration. Valid values include GYRO_DLPF_196_6HZ, GYRO_DLPF_151_8HZ, GYRO_DLPF_119_5HZ, GYRO_DLPF_51_2HZ, GYRO_DLPF_23_9HZ, GYRO_DLPF_11_6HZ, GYRO_DLPF_5_7HZ, and GYRO_DLPF_361_4HZ. |

**Returns**

A string representing the gyroscope DLPF configuration. The possible return values are "196.6Hz", "151.8Hz", "119.5Hz", "51.2Hz", "23.9Hz", "11.6Hz", "5.7Hz", and "361.4Hz". If an invalid gyro_dlpf_config value is provided, the function returns "<invalid gyroscope DLPF config>".

#### 5.4.3.5 gyro_scale_factor()

```
float icm20948::gyro_scale_factor (
            gyro_scale scale)
```

Calculates the scale factor for gyroscope measurements.

This function determines the scale factor for the gyroscope based on the selected full-scale range. The scale factor is used to convert raw sensor readings into meaningful angular velocity values in degrees per second (DPS).

**Parameters**

| | |
|---|---|
| *scale* | The full-scale range setting for the gyroscope, defined by the gyro_scale enumeration. Valid values include GYRO_250DPS, GYRO_500DPS, GYRO_1000DPS, and GYRO_2000DPS. |

**Returns**

The scale factor corresponding to the specified gyroscope full-scale range. The scale factor is expressed as the reciprocal of the full-scale measurement range in terms of DPS.

**Exceptions**

| | |
|---|---|
| *std::invalid_argument* | If an invalid `gyro_scale` value is provided. |

#### 5.4.3.6 gyro_scale_to_str()

```
std::string icm20948::gyro_scale_to_str (
            gyro_scale scale)
```

Converts a gyroscope scale value to its corresponding string representation.

This function maps a `gyro_scale` enumeration value to a human-readable string that represents the gyroscope's full-scale range setting.

**Parameters**

| | |
|---|---|
| *scale* | The full-scale range setting for the gyroscope, defined by the `gyro_scale` enumeration. Valid values include GYRO_250DPS, GYRO_500DPS, GYRO_1000DPS, and GYRO_2000DPS. |

**Returns**

A string representing the gyroscope scale. The possible return values are "250DPS", "500DPS", "1000DPS", and "2000DPS". If an invalid `gyro_scale` value is provided, the function returns "<invalid gyroscope scale>".

#### 5.4.3.7 magn_mode_to_str()

```
std::string icm20948::magn_mode_to_str (
            magn_mode mode)
```

Converts a magnetometer operation mode value to its corresponding string representation.

This function maps a `magn_mode` enumeration value to a human-readable string that represents the magnetometer's operation mode or measurement frequency.

**Parameters**

| | |
|---|---|
| *mode* | The operation mode of the magnetometer, defined by the `magn_mode` enumeration. Valid values include MAGN_SHUTDOWN, MAGN_SINGLE, MAGN_10HZ, MAGN_20HZ, MAGN_50HZ, MAGN_100HZ, and MAGN_SELF_TEST. |

**Returns**

A string representing the magnetometer mode. The possible return values are "Shutdown", "Single", "10Hz", "20Hz", "50Hz", "100Hz", and "Self-test". If an invalid `magn_mode` value is provided, the function returns "<invalid magnetometer mode>".

## 5.5 IMUMathsName Namespace Reference

### Classes

- class IMUMaths

## 5.6 PlayAudioName Namespace Reference

### Classes

- class PlayAudio

# Chapter 6

# Class Documentation

## 6.1 icm20948::accel_settings Struct Reference

Configuration settings for an accelerometer sensor.

`#include <icm20948_utils.hpp>`

### Public Member Functions

- accel_settings (uint16_t sample_rate_div=0, accel_scale scale=ACCEL_2G, bool dlpf_enable=true, accel_dlpf_config dlpf_config=ACCEL_DLPF_246HZ)

### Public Attributes

- uint16_t sample_rate_div
- accel_scale scale
- bool dlpf_enable
- accel_dlpf_config dlpf_config

### 6.1.1 Detailed Description

Configuration settings for an accelerometer sensor.

This structure defines the configuration parameters for the accelerometer, including the sample rate, full-scale range, and digital low-pass filter (DLPF) settings.

Members:

- uint16_t `sample_rate_div`: The sample rate divider, which determines the inner sample data rate of the accelerometer. The value should be in the range [0, 4095]. The formula for calculating the inner sample rate is: 1.125kHz / sample_rate_div
- accel_scale `scale`: The full-scale range of the accelerometer measurements. This defines the maximum acceleration (in g) that the sensor can measure. The available options are defined in the `accel_scale` enumeration.

- bool `dlpf_enable`: A flag indicating whether the Digital Low-Pass Filter (DLPF) is enabled. If true, the DLPF is active and the cutoff frequency is determined by `dlpf_config`.
- accel_dlpf_config `dlpf_config`: The configuration settings for the DLPF, determining the filter's cutoff frequency. The available options are defined in the `accel_dlpf_config` enumeration.

Constructor:

- accel_settings(uint16_t sample_rate_div = 0, accel_scale scale = ACCEL_2G, bool dlpf_enable = true, accel_dlpf_config dlpf_config = ACCEL_DLPF_246HZ) Initializes the structure with default values or specified parameters.

### 6.1.2 Constructor & Destructor Documentation

#### 6.1.2.1 accel_settings()

```
icm20948::accel_settings::accel_settings (
            uint16_t sample_rate_div = 0,
            accel_scale scale = ACCEL_2G,
            bool dlpf_enable = true,
            accel_dlpf_config dlpf_config = ACCEL_DLPF_246HZ)  [inline]
```

### 6.1.3 Member Data Documentation

#### 6.1.3.1 dlpf_config

```
accel_dlpf_config icm20948::accel_settings::dlpf_config
```

#### 6.1.3.2 dlpf_enable

```
bool icm20948::accel_settings::dlpf_enable
```

#### 6.1.3.3 sample_rate_div

```
uint16_t icm20948::accel_settings::sample_rate_div
```

#### 6.1.3.4 scale

```
accel_scale icm20948::accel_settings::scale
```

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

- src/libs/I2C/include/icm20948_utils.hpp

## 6.2 AiffUtilities Struct Reference

`#include <AudioFile.h>`

### Static Public Member Functions

- static double decodeAiffSampleRate (const uint8_t *bytes)
- static void encodeAiffSampleRate (double sampleRate, uint8_t *bytes)

### 6.2.1 Member Function Documentation

#### 6.2.1.1 decodeAiffSampleRate()

```
double AiffUtilities::decodeAiffSampleRate (
            const uint8_t * bytes)  [inline], [static]
```

Decode an 80-bit (10 byte) sample rate to a double

#### 6.2.1.2 encodeAiffSampleRate()

```
void AiffUtilities::encodeAiffSampleRate (
            double sampleRate,
            uint8_t * bytes)  [inline], [static]
```

Encode a double as an 80-bit (10-byte) sample rate

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

- src/libs/ALSAPlayer/include/AudioFile.h

## 6.3 AudioFile< T > Class Template Reference

`#include <AudioFile.h>`

### Public Types

- typedef std::vector< std::vector< T > > AudioBuffer

**Public Member Functions**

- AudioFile ()
- AudioFile (const std::string &filePath)
- bool load (const std::string &filePath)
- bool save (const std::string &filePath, AudioFileFormat format=AudioFileFormat::Wave)
- bool loadFromMemory (const std::vector< uint8_t > &fileData)
- uint32_t getSampleRate () const
- int getNumChannels () const
- bool isMono () const
- bool isStereo () const
- int getBitDepth () const
- int getNumSamplesPerChannel () const
- double getLengthInSeconds () const
- void printSummary () const
- bool setAudioBuffer (const AudioBuffer &newBuffer)
- void setAudioBufferSize (const int numChannels, const int numSamples)
- void setNumSamplesPerChannel (const int numSamples)
- void setNumChannels (const int numChannels)
- void setBitDepth (const int numBitsPerSample)
- void setSampleRate (const uint32_t newSampleRate)
- void shouldLogErrorsToConsole (bool logErrors)

**Public Attributes**

- AudioBuffer samples
- std::string iXMLChunk

### 6.3.1 Member Typedef Documentation

#### 6.3.1.1 AudioBuffer

```
template<class T>
typedef std::vector<std::vector<T> > AudioFile< T >::AudioBuffer
```

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 AudioFile() [1/2]

```
template<class T>
AudioFile< T >::AudioFile ()
```

Constructor

#### 6.3.2.2 AudioFile() [2/2]

```
template<class T>
AudioFile< T >::AudioFile (
            const std::string & filePath)
```

Constructor, using a given file path to load a file

## 6.3.3 Member Function Documentation

### 6.3.3.1 getBitDepth()

```
template<class T>
int AudioFile< T >::getBitDepth () const
```

@Returns the bit depth of each sample

### 6.3.3.2 getLengthInSeconds()

```
template<class T>
double AudioFile< T >::getLengthInSeconds () const
```

@Returns the length in seconds of the audio file based on the number of samples and sample rate

### 6.3.3.3 getNumChannels()

```
template<class T>
int AudioFile< T >::getNumChannels () const
```

@Returns the number of audio channels in the buffer

### 6.3.3.4 getNumSamplesPerChannel()

```
template<class T>
int AudioFile< T >::getNumSamplesPerChannel () const
```

@Returns the number of samples per channel

### 6.3.3.5 getSampleRate()

```
template<class T>
uint32_t AudioFile< T >::getSampleRate () const
```

@Returns the sample rate

### 6.3.3.6 isMono()

```
template<class T>
bool AudioFile< T >::isMono () const
```

@Returns true if the audio file is mono

#### 6.3.3.7 isStereo()

```
template<class T>
bool AudioFile< T >::isStereo () const
```

@Returns true if the audio file is stereo

#### 6.3.3.8 load()

```
template<class T>
bool AudioFile< T >::load (
            const std::string & filePath)
```

Loads an audio file from a given file path. @Returns true if the file was successfully loaded

#### 6.3.3.9 loadFromMemory()

```
template<class T>
bool AudioFile< T >::loadFromMemory (
            const std::vector< uint8_t > & fileData)
```

Loads an audio file from data in memory

#### 6.3.3.10 printSummary()

```
template<class T>
void AudioFile< T >::printSummary () const
```

Prints a summary of the audio file to the console

#### 6.3.3.11 save()

```
template<class T>
bool AudioFile< T >::save (
            const std::string & filePath,
            AudioFileFormat format = AudioFileFormat::Wave)
```

Saves an audio file to a given file path. @Returns true if the file was successfully saved

#### 6.3.3.12 setAudioBuffer()

```
template<class T>
bool AudioFile< T >::setAudioBuffer (
            const AudioBuffer & newBuffer)
```

Set the audio buffer for this AudioFile by copying samples from another buffer. @Returns true if the buffer was copied successfully.

#### 6.3.3.13 setAudioBufferSize()

```
template<class T>
void AudioFile< T >::setAudioBufferSize (
            const int numChannels,
            const int numSamples)
```

Sets the audio buffer to a given number of channels and number of samples per channel. This will try to preserve the existing audio, adding zeros to any new channels or new samples in a given channel.

#### 6.3.3.14 setBitDepth()

```
template<class T>
void AudioFile< T >::setBitDepth (
            const int numBitsPerSample)
```

Sets the bit depth for the audio file. If you use the save() function, this bit depth rate will be used

#### 6.3.3.15 setNumChannels()

```
template<class T>
void AudioFile< T >::setNumChannels (
            const int numChannels)
```

Sets the number of channels. New channels will have the correct number of samples and be initialised to zero

#### 6.3.3.16 setNumSamplesPerChannel()

```
template<class T>
void AudioFile< T >::setNumSamplesPerChannel (
            const int numSamples)
```

Sets the number of samples per channel in the audio buffer. This will try to preserve the existing audio, adding zeros to new samples in a given channel if the number of samples is increased.

#### 6.3.3.17 setSampleRate()

```
template<class T>
void AudioFile< T >::setSampleRate (
            const uint32_t newSampleRate)
```

Sets the sample rate for the audio file. If you use the save() function, this sample rate will be used

#### 6.3.3.18 shouldLogErrorsToConsole()

```
template<class T>
void AudioFile< T >::shouldLogErrorsToConsole (
            bool logErrors)
```

Sets whether the library should log error messages to the console. By default this is true

### 6.3.4 Member Data Documentation

#### 6.3.4.1 iXMLChunk

```
template<class T>
std::string AudioFile< T >::iXMLChunk
```

An optional iXML chunk that can be added to the AudioFile.

#### 6.3.4.2 samples

```
template<class T>
AudioBuffer AudioFile< T >::samples
```

A vector of vectors holding the audio samples for the AudioFile. You can access the samples by channel and then by sample index, i.e:

```
samples[channel][sampleIndex]
```

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

- src/libs/ALSAPlayer/include/AudioFile.h

## 6.4 AudioLib::AudioLib Class Reference

```
#include <AudioLib.hpp>
```

#### Public Member Functions

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

### 6.4.1 Constructor & Destructor Documentation

#### 6.4.1.1 AudioLib()

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

#### 6.4.1.2 ~AudioLib()

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

### 6.4.2 Member Function Documentation

#### 6.4.2.1 PlayAudioTerminal()

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

#### 6.4.2.2 PlayFile()

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

#### 6.4.2.3 PlaySound()

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

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

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

## 6.5 AudioPlayerName::AudioPlayer Class Reference

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

#### 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 ()
- bool playFile (const std::string &fileKey)
- void close ()
- ~AudioPlayer ()

#### Public Attributes

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

### 6.5.1 Constructor & Destructor Documentation

#### 6.5.1.1 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 = 32,
            const std::vector< std::string > & filesToConvert = {"src/libs/ALSAPlayer/include/CrashCymbal.wa
[inline]
```

#### 6.5.1.2 ~AudioPlayer()

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

### 6.5.2 Member Function Documentation

#### 6.5.2.1 close()

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

#### 6.5.2.2 open()

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

#### 6.5.2.3 playFile()

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

### 6.5.3 Member Data Documentation

#### 6.5.3.1 audioBuffer

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

#### 6.5.3.2 CancelPlayback

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

#### 6.5.3.3 fileBuffers

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

#### 6.5.3.4 StopMixingThread

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

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

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

## 6.6 AudioSampleConverter< T > Struct Template Reference

```
#include <AudioFile.h>
```

### Static Public Member Functions

- static T signedByteToSample (int8_t sample)
- static int8_t sampleToSignedByte (T sample)
- static T unsignedByteToSample (uint8_t sample)
- static uint8_t sampleToUnsignedByte (T sample)
- static T sixteenBitIntToSample (int16_t sample)
- static int16_t sampleToSixteenBitInt (T sample)
- static T twentyFourBitIntToSample (int32_t sample)
- static int32_t sampleToTwentyFourBitInt (T sample)
- static T thirtyTwoBitIntToSample (int32_t sample)
- static int32_t sampleToThirtyTwoBitInt (T sample)
- static T clamp (T v1, T minValue, T maxValue)

### 6.6.1 Member Function Documentation

#### 6.6.1.1 clamp()

```
template<class T>
T AudioSampleConverter< T >::clamp (
            T v1,
            T minValue,
            T maxValue)  [static]
```

Helper clamp function to enforce ranges

#### 6.6.1.2 sampleToSignedByte()

```
template<class T>
int8_t AudioSampleConverter< T >::sampleToSignedByte (
            T sample)  [static]
```

Convert an audio sample to an signed 8-bit representation

#### 6.6.1.3 sampleToSixteenBitInt()

```
template<class T>
int16_t AudioSampleConverter< T >::sampleToSixteenBitInt (
            T sample)  [static]
```

Convert a an audio sample to a 16-bit integer

#### 6.6.1.4 sampleToThirtyTwoBitInt()

```
template<class T>
int32_t AudioSampleConverter< T >::sampleToThirtyTwoBitInt (
            T sample)  [static]
```

Convert a an audio sample to a 32-bit signed integer

#### 6.6.1.5 sampleToTwentyFourBitInt()

```
template<class T>
int32_t AudioSampleConverter< T >::sampleToTwentyFourBitInt (
            T sample)  [static]
```

Convert a an audio sample to a 24-bit value (in a 32-bit integer)

#### 6.6.1.6 sampleToUnsignedByte()

```
template<class T>
uint8_t AudioSampleConverter< T >::sampleToUnsignedByte (
            T sample)  [static]
```

Convert an audio sample to an unsigned 8-bit representation

#### 6.6.1.7 signedByteToSample()

```
template<class T>
T AudioSampleConverter< T >::signedByteToSample (
            int8_t sample)  [static]
```

Convert a signed 8-bit integer to an audio sample

#### 6.6.1.8 sixteenBitIntToSample()

```
template<class T>
T AudioSampleConverter< T >::sixteenBitIntToSample (
            int16_t sample)  [static]
```

Convert a 16-bit integer to an audio sample

#### 6.6.1.9 thirtyTwoBitIntToSample()

```
template<class T>
T AudioSampleConverter< T >::thirtyTwoBitIntToSample (
            int32_t sample)  [static]
```

Convert a 32-bit signed integer to an audio sample

#### 6.6.1.10 twentyFourBitIntToSample()

```
template<class T>
T AudioSampleConverter< T >::twentyFourBitIntToSample (
            int32_t sample)  [static]
```

Convert a 24-bit value (int a 32-bit int) to an audio sample

#### 6.6.1.11 unsignedByteToSample()

```
template<class T>
T AudioSampleConverter< T >::unsignedByteToSample (
            uint8_t sample)  [static]
```

Convert an unsigned 8-bit integer to an audio sample

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

- src/libs/ALSAPlayer/include/AudioFile.h

## 6.7 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 ()

  *Changes a boolean to end the worker.*
- 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

## 6.7.1 Constructor & Destructor Documentation

### 6.7.1.1 GPIOClass()

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

## 6.7.2 Member Function Documentation

### 6.7.2.1 GPIOStop()

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

Changes a boolean to end the worker.

### 6.7.2.2 IMUMathsCallback()

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

### 6.7.2.3 SetCallback()

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

### 6.7.2.4 Worker()

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

Event driven worker reading data when HIGH seen on GPIO.

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

#### 6.7.2.5 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.

### 6.7.3 Member Data Documentation

#### 6.7.3.1 callback

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

#### 6.7.3.2 CallbackFunction

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

#### 6.7.3.3 Maths

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

#### 6.7.3.4 sensor

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

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

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

## 6.8 icm20948::gyro_settings Struct Reference

Configuration settings for a gyroscope sensor.

```
#include <icm20948_utils.hpp>
```

#### Public Member Functions

- gyro_settings (uint8_t sample_rate_div=0, gyro_scale scale=GYRO_250DPS, bool dlpf_enable=true, gyro_dlpf_config dlpf_config=GYRO_DLPF_196_6HZ)

**Public Attributes**

- uint8_t sample_rate_div
- gyro_scale scale
- bool dlpf_enable
- gyro_dlpf_config dlpf_config

### 6.8.1 Detailed Description

Configuration settings for a gyroscope sensor.

This structure defines the configuration parameters for a gyroscope sensor, including the sample rate, full-scale range, and Digital Low-Pass Filter (DLPF) settings.

Members:

- uint8_t sample_rate_div: The sample rate divider, which determines the inner sample data rate of the gyroscope. The value should be in the range [0, 250]. The formula for calculating the inner sample rate is: 1.1kHz / sample_rate_div
- gyro_scale scale: The full-scale range of the gyroscope measurements. This defines the maximum angular velocity (in degrees per second) that the sensor can measure. The available options are defined in the `gyro_scale` enumeration.
- bool dlpf_enable: A flag indicating whether the Digital Low-Pass Filter (DLPF) is enabled. If true, the DLPF is active and the cutoff frequency is determined by `dlpf_config`.
- gyro_dlpf_config dlpf_config: The configuration settings for the DLPF, determining the filter's cutoff frequency. The available options are defined in the `gyro_dlpf_config` enumeration.

Constructor:

- gyro_settings(uint8_t sample_rate_div = 0, gyro_scale scale = GYRO_250DPS, bool dlpf_enable = true, gyro_dlpf_config dlpf_config = GYRO_DLPF_196_6HZ) Initializes the structure with default values or specified parameters.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 gyro_settings()

```
icm20948::gyro_settings::gyro_settings (
            uint8_t sample_rate_div = 0,
            gyro_scale scale = GYRO_250DPS,
            bool dlpf_enable = true,
            gyro_dlpf_config dlpf_config = GYRO_DLPF_196_6HZ)  [inline]
```

### 6.8.3 Member Data Documentation

#### 6.8.3.1 dlpf_config

```
gyro_dlpf_config icm20948::gyro_settings::dlpf_config
```

#### 6.8.3.2 dlpf_enable

```
bool icm20948::gyro_settings::dlpf_enable
```

#### 6.8.3.3 sample_rate_div

```
uint8_t icm20948::gyro_settings::sample_rate_div
```

#### 6.8.3.4 scale

```
gyro_scale icm20948::gyro_settings::scale
```

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

- src/libs/I2C/include/icm20948_utils.hpp

## 6.9 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

### 6.9.1 Constructor & Destructor Documentation

#### 6.9.1.1 ICM20948_I2C()

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

### 6.9.2 Member Function Documentation

#### 6.9.2.1 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

#### 6.9.2.2 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

#### 6.9.2.3 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.

#### 6.9.2.4 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.

#### 6.9.2.5 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.

#### 6.9.2.6 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.

#### 6.9.2.7 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.

#### 6.9.2.8 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.

### 6.9.3 Member Data Documentation

#### 6.9.3.1 accel

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

#### 6.9.3.2 gyro

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

#### 6.9.3.3 magn

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

#### 6.9.3.4 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

## 6.10 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

### 6.10.1 Constructor & Destructor Documentation

#### 6.10.1.1 IMUMaths()

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

Constructs an object with access to the audio player.

**Parameters**

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

### 6.10.2 Member Function Documentation

#### 6.10.2.1 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* | |

#### 6.10.2.2 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 |

### 6.10.3 Member Data Documentation

#### 6.10.3.1 Audio

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

#### 6.10.3.2 Counter

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

#### 6.10.3.3 LastFilePlayed

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

#### 6.10.3.4 Pause

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

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

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

# 6.11 icm20948::magn_settings Struct Reference

Configuration settings for a magnetometer sensor.

`#include <icm20948_utils.hpp>`

### Public Member Functions

- magn_settings (magn_mode mode=MAGN_100HZ)

### Public Attributes

- magn_mode mode

## 6.11.1 Detailed Description

Configuration settings for a magnetometer sensor.

This structure defines the configuration parameters for a magnetometer sensor, including the operational mode. The mode determines how the sensor operates, including its measurement rate and diagnostic functions.

Members:

- magn_mode mode: The operation mode of the magnetometer sensor. This determines the sensor's measurement frequency or operational state. The available options are defined in the `magn_mode` enumeration.

Constructor:

- magn_settings(magn_mode mode = MAGN_100HZ) Initializes the structure with the specified mode or a default mode of MAGN_100HZ.

## 6.11.2 Constructor & Destructor Documentation

### 6.11.2.1 magn_settings()

```
icm20948::magn_settings::magn_settings (
            magn_mode mode = MAGN_100HZ)  [inline]
```

## 6.11.3 Member Data Documentation

### 6.11.3.1 mode

```
magn_mode icm20948::magn_settings::mode
```

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

- src/libs/I2C/include/icm20948_utils.hpp

## 6.12 PlayAudioName::PlayAudio Class Reference

`#include <PlayAudio.hpp>`

### Static Public Member Functions

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

### 6.12.1 Member Function Documentation

#### 6.12.1.1 PlayCymbal()

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

#### 6.12.1.2 PlayHighTom()

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

#### 6.12.1.3 PlaySnare()

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

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

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

## 6.13 icm20948::settings Struct Reference

Aggregated configuration settings for sensor modules.

`#include <icm20948_utils.hpp>`

Collaboration diagram for icm20948::settings:

### Public Member Functions

- settings (accel_settings accel=accel_settings(), gyro_settings gyro=gyro_settings(), magn_settings magn=magn_settings())
- settings (YAML::Node config_file_node)

  *Constructs a `settings` object from a YAML configuration node.*

**Public Attributes**

- accel_settings accel
- gyro_settings gyro
- magn_settings magn

### 6.13.1 Detailed Description

Aggregated configuration settings for sensor modules.

This structure contains the configuration parameters for multiple sensor modules, including accelerometer, gyroscope, and magnetometer settings. It allows for centralized management of all sensor settings in a single structure.

Members:

- accel_settings `accel`: Configuration settings for the accelerometer sensor, defining parameters such as inner sample rate, full-scale range, and Digital Low-Pass Filter (DLPF) settings.
- gyro_settings `gyro`: Configuration settings for the gyroscope sensor, including inner sample rate, full-scale range, and Digital Low-Pass Filter (DLPF) settings.
- magn_settings `magn`: Configuration settings for the magnetometer sensor, specifying the operation mode and measurement frequency.

Constructors:

- settings(accel_settings accel = accel_settings(), gyro_settings gyro = gyro_settings(), magn_settings magn = magn_settings()) Initializes the structure with specified or default settings for accelerometer, gyroscope, and magnetometer.
- settings(YAML::Node config_file_node) Initializes the structure with settings loaded from a YAML configuration file node.

### 6.13.2 Constructor & Destructor Documentation

#### 6.13.2.1 settings() [1/2]

```
icm20948::settings::settings (
            accel_settings accel = accel_settings(),
            gyro_settings gyro = gyro_settings(),
            magn_settings magn = magn_settings())  [inline]
```

#### 6.13.2.2 settings() [2/2]

```
icm20948::settings::settings (
            YAML::Node config_file_node)
```

Constructs a `settings` object from a YAML configuration node.

This constructor initializes the `settings` structure by parsing a YAML configuration node. It sets up the accelerometer, gyroscope, and magnetometer settings based on the values provided in the YAML file.

**Parameters**

| | |
|---|---|
| *config_file_node* | A YAML node containing configuration data for the accelerometer, gyroscope, and magnetometer. The node should include sub-nodes for each sensor type with relevant settings such as sample rate, scale, and filter configurations. |

**Exceptions**

| | |
|---|---|
| *std::runtime_error* | If an invalid mode is specified for the magnetometer in the YAML configuration. |

### 6.13.3 Member Data Documentation

#### 6.13.3.1 accel

```
accel_settings icm20948::settings::accel
```

#### 6.13.3.2 gyro

```
gyro_settings icm20948::settings::gyro
```

#### 6.13.3.3 magn

```
magn_settings icm20948::settings::magn
```

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

- src/libs/I2C/include/icm20948_utils.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:

## 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     class AudioPlayer{
00016         public:
00017         std::vector<int32_t> audioBuffer;
00018         std::unordered_map<std::string, std::vector<int32_t» fileBuffers;
00019
00020         bool StopMixingThread;
00021         bool CancelPlayback = true;
00022
00023
00024         AudioPlayer(const std::string& device="default",
00025             unsigned int rate = 44100,
00026             unsigned int ch = 2,
00027             snd_pcm_format_t fmt = SND_PCM_FORMAT_S16_LE,
00028             snd_pcm_uframes_t frames = 32,
00029             const std::vector<std::string>& filesToConvert =
      {"src/libs/ALSAPlayer/include/CrashCymbal.wav",
00030
      "src/libs/ALSAPlayer/include/HighTom.wav",
00031
      "src/libs/ALSAPlayer/include/SnareDrum.wav"})
00032         : deviceName(device), sampleRate(rate), channels(ch),
00033         format(fmt), framesPerPeriod(frames), handle(nullptr)
00034         {
00035             //Convert files to audio buffers here I think
00036             if (!filesToConvert.empty()){
00037                 ConvertFiles(filesToConvert);
00038             }
00039         }
00040
00041         bool open(){
00042             int rc = snd_pcm_open(&handle, deviceName.c_str(), SND_PCM_STREAM_PLAYBACK,0);
00043             if (rc < 0){
00044                 std::cerr « "Unable to open PCM devices: " « snd_strerror(rc) « std::endl;
00045                 return false;
00046             }
00047
00048             snd_pcm_hw_params_t* params;
00049             snd_pcm_hw_params_alloca(&params);
00050             snd_pcm_hw_params_any(handle, params);
00051             snd_pcm_hw_params_set_access(handle, params, SND_PCM_ACCESS_RW_INTERLEAVED);
00052             snd_pcm_hw_params_set_format(handle, params, format);
00053             snd_pcm_hw_params_set_channels(handle, params, channels);
00054
00055             unsigned int rate_near = sampleRate;
00056             snd_pcm_hw_params_set_rate_near(handle, params, &rate_near,0);
00057             snd_pcm_hw_params_set_period_size_near(handle, params, &framesPerPeriod, 0);
00058
```

```
00059             rc = snd_pcm_hw_params(handle, params);
00060             if (rc <0) {
00061                 std::cerr « "Unable to set HW parameters: " « snd_strerror(rc) « std::endl;
00062                 return false;
00063             }
00064 
00065             snd_pcm_hw_params_get_period_size(params, &framesPerPeriod,0);
00066             return true;
00067         }
00068 
00069         bool playFile(const std::string& fileKey) {
00070 
00071             CancelPlayback = true;
00072 
00073             if(handle) {
00074                 snd_pcm_drop(handle);
00075                 snd_pcm_prepare(handle);
00076             }
00077 
00078             CancelPlayback = false;
00079 
00080             if (!handle) {
00081                 std::cerr « "Device not open. Call open() first.\n";
00082                 return false;
00083             }
00084             if (fileBuffers.find(fileKey) == fileBuffers.end()) {
00085                 std::cerr « "Audio buffer not found for file: " « fileKey « "\n";
00086                 return false;
00087             }
00088 
00089             const std::vector<int32_t>& buffer = fileBuffers[fileKey];
00090             size_t totalFrames = buffer.size() / channels;
00091             size_t offset = 0;
00092             int rc = 0;
00093 
00094             if (CancelPlayback){
00095                 std::cerr « "[DEBUG] Playback cancelled." « std::endl;
00096                 return false;
00097             }
00098 
00099 
00100             while (offset < totalFrames) {
00101                 snd_pcm_uframes_t framesToWrite = framesPerPeriod;
00102                 if (offset + framesPerPeriod > totalFrames)
00103                     framesToWrite = totalFrames - offset;
00104                 rc = snd_pcm_writei(handle, buffer.data() + offset * channels, framesToWrite);
00105                 if (rc == -EPIPE) {
00106                     std::cerr « "Underrun occurred\n";
00107                     snd_pcm_prepare(handle);
00108                 } else if (rc < 0) {
00109                     std::cerr « "Error from writei: " « snd_strerror(rc) « "\n";
00110                     return false;
00111                 } else if (static_cast<snd_pcm_uframes_t>(rc) != framesToWrite) {
00112                     std::cerr « "Short write, wrote " « rc « " frames\n";
00113                 } else {
00114                     offset += rc;
00115                 }
00116             }
00117             snd_pcm_drain(handle);
00118             snd_pcm_prepare(handle);
00119             return true;
00120         }
00121 
00122         void close() {
00123             if (handle) {
00124                 snd_pcm_close(handle);
00125                 handle = nullptr;
00126             }
00127         }
00128 
00129         ~AudioPlayer() {
00130             close();
00131         }
00132 
00133 
00134         private:
00135         std::string deviceName;
00136         unsigned int sampleRate;
00137         unsigned int channels;
00138         snd_pcm_format_t format;
00139         snd_pcm_uframes_t framesPerPeriod;
00140         snd_pcm_t* handle;
00141 
00142         std::condition_variable MixCV;
00143         std::mutex MixCVMutex;
00144 
00145         struct ActiveSound {
```

```
00146             std::vector<int32_t>* buffer;
00147             size_t position;
00148         };
00149 
00150         std::vector<ActiveSound> ActiveSounds;
00151         std::mutex ActiveMutex;
00152 
00153 
00154         template<typename T>
00155         void printVector(const std::vector<T>& vec) {
00156         for (const auto& el : vec) {
00157             std::cout « el « " ";
00158         }
00159         std::cout « std::endl;
00160         }
00161 
00162         void ConvertFiles(const std::vector<std::string>& filePaths) {
00163             std::vector<int32_t> result;
00164 
00165             for (const auto& path : filePaths) {
00166                 AudioFile<int32_t> file;
00167                 if (!file.load(path)) {
00168                     std::cerr « "Error loading file: " « path « std::endl;
00169                     continue;
00170                 }
00171 
00172                 int fileChannels = file.getNumChannels();
00173                 int ChannelSamples = file.getNumSamplesPerChannel();
00174 
00175                 std::vector<int32_t> interleaved;
00176                 interleaved.reserve(ChannelSamples * fileChannels);
00177                 for (int i=0; i < ChannelSamples; ++i){
00178                     for (int ch = 0; ch < fileChannels; ++ch){
00179                         interleaved.push_back(file.samples[ch][i]);
00180 
00181                     }
00182 
00183 
00184                 }
00185                 //printVector(interleaved);
00186                 fileBuffers[path] = std::move(interleaved);
00187 
00188             }
00189         }
00190 
00191 
00192     };
00193 }
00194 
00195 
00196 
00197 #endif
```

## 7.6 src/libs/ALSAPlayer/include/AudioFile.h File Reference

```
#include <iostream>
#include <vector>
#include <cassert>
#include <string>
#include <cstring>
#include <fstream>
#include <unordered_map>
#include <iterator>
#include <algorithm>
#include <limits>
#include <cstdint>
#include <cmath>
#include <array>
```

Include dependency graph for AudioFile.h: This graph shows which files directly or indirectly include this file:

### Classes

- class AudioFile< T >

- struct AudioSampleConverter< T >
- struct AiffUtilities

### Enumerations

- enum class AudioFileFormat { Error , NotLoaded , Wave , Aiff }
- enum WavAudioFormat {
  PCM = 0x0001 , IEEEFloat = 0x0003 , ALaw = 0x0006 , MULaw = 0x0007 ,
  Extensible = 0xFFFE }
- enum AIFFAudioFormat { Uncompressed , Compressed , Error }
- enum SampleLimit {
  SignedInt16_Min = -32768 , SignedInt16_Max = 32767 , UnsignedInt16_Min = 0 , UnsignedInt16_Max = 65535 ,
  SignedInt24_Min = -8388608 , SignedInt24_Max = 8388607 , UnsignedInt24_Min = 0 , UnsignedInt24_Max = 16777215 }

### Functions

- template<typename SignedType>
  std::make_unsigned< SignedType >::type convertSignedToUnsigned (SignedType signedValue)

## 7.6.1 Detailed Description

**Author**

Adam Stark

**Copyright**

Copyright (C) 2017 Adam Stark

This file is part of the 'AudioFile' library

MIT License

Copyright (c) 2017 Adam Stark

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 7.6.2 Enumeration Type Documentation

### 7.6.2.1 AIFFAudioFormat

```
enum AIFFAudioFormat
```

**Enumerator**

| | |
|---|---|
| Uncompressed | |
| Compressed | |
| Error | |

#### 7.6.2.2 AudioFileFormat

```
enum class AudioFileFormat  [strong]
```

The different types of audio file, plus some other types to indicate a failure to load a file, or that one hasn't been loaded yet

**Enumerator**

| | |
|---|---|
| Error | |
| NotLoaded | |
| Wave | |
| Aiff | |

#### 7.6.2.3 SampleLimit

```
enum SampleLimit
```

**Enumerator**

| | |
|---|---|
| SignedInt16_Min | |
| SignedInt16_Max | |
| UnsignedInt16_Min | |
| UnsignedInt16_Max | |
| SignedInt24_Min | |
| SignedInt24_Max | |
| UnsignedInt24_Min | |
| UnsignedInt24_Max | |

#### 7.6.2.4 WavAudioFormat

```
enum WavAudioFormat
```

**Enumerator**

| | |
|---|---|
| PCM | |
| IEEEFloat | |
| ALaw | |
| MULaw | |
| Extensible | |

### 7.6.3 Function Documentation

#### 7.6.3.1 convertSignedToUnsigned()

```
template<typename SignedType>
std::make_unsigned< SignedType >::type convertSignedToUnsigned (
              SignedType signedValue)
```

## 7.7 AudioFile.h

Go to the documentation of this file.

```
00001 //=============================================================================
00029 //=============================================================================
00030
00031 #ifndef _AS_AudioFile_h
00032 #define _AS_AudioFile_h
00033
00034 #if defined (_MSC_VER)
00035 #undef max
00036 #undef min
00037 #define NOMINMAX
00038 #endif
00039
00040 #include <iostream>
00041 #include <vector>
00042 #include <cassert>
00043 #include <string>
00044 #include <cstring>
00045 #include <fstream>
00046 #include <unordered_map>
00047 #include <iterator>
00048 #include <algorithm>
00049 #include <limits>
00050 #include <cstdint>
00051 #include <cmath>
00052 #include <array>
00053
00054 // disable some warnings on Windows
00055 #if defined (_MSC_VER)
00056     __pragma(warning (push))
00057     __pragma(warning (disable : 4244))
00058     __pragma(warning (disable : 4457))
00059     __pragma(warning (disable : 4458))
00060     __pragma(warning (disable : 4389))
00061     __pragma(warning (disable : 4996))
00062 #elif defined (__GNUC__)
00063     _Pragma("GCC diagnostic push")
00064     _Pragma("GCC diagnostic ignored \"-Wconversion\"")
00065     _Pragma("GCC diagnostic ignored \"-Wsign-compare\"")
00066     _Pragma("GCC diagnostic ignored \"-Wshadow\"")
00067 #endif
00068
00069 //=============================================================
00074 enum class AudioFileFormat
00075 {
00076     Error,
00077     NotLoaded,
00078     Wave,
00079     Aiff
00080 };
00081
00082 //=============================================================
00083 template <class T>
00084 class AudioFile
00085 {
00086 public:
00087
00088     //=========================================================
00089     typedef std::vector<std::vector<T> > AudioBuffer;
00090
00091     //=========================================================
00093     AudioFile();
00094
00096     AudioFile (const std::string& filePath);
00097
00098     //=========================================================
```

```
00102     bool load (const std::string& filePath);
00103
00107     bool save (const std::string& filePath, AudioFileFormat format = AudioFileFormat::Wave);
00108
00109     //=============================================================
00111     bool loadFromMemory (const std::vector<uint8_t>& fileData);
00112
00113     //=============================================================
00115     uint32_t getSampleRate() const;
00116
00118     int getNumChannels() const;
00119
00121     bool isMono() const;
00122
00124     bool isStereo() const;
00125
00127     int getBitDepth() const;
00128
00130     int getNumSamplesPerChannel() const;
00131
00133     double getLengthInSeconds() const;
00134
00136     void printSummary() const;
00137
00138     //=============================================================
00139
00143     bool setAudioBuffer (const AudioBuffer& newBuffer);
00144
00148     void setAudioBufferSize (const int numChannels, const int numSamples);
00149
00153     void setNumSamplesPerChannel (const int numSamples);
00154
00156     void setNumChannels (const int numChannels);
00157
00159     void setBitDepth (const int numBitsPerSample);
00160
00162     void setSampleRate (const uint32_t newSampleRate);
00163
00164     //=============================================================
00166     void shouldLogErrorsToConsole (bool logErrors);
00167
00168     //=============================================================
00174     AudioBuffer samples;
00175
00176     //=============================================================
00179     std::string iXMLChunk;
00180
00181 private:
00182
00183     //=============================================================
00184     enum class Endianness
00185     {
00186         LittleEndian,
00187         BigEndian
00188     };
00189
00190     //=============================================================
00191     bool decodeWaveFile (const std::vector<uint8_t>& fileData);
00192     bool decodeAiffFile (const std::vector<uint8_t>& fileData);
00193
00194     //=============================================================
00195     bool saveToWaveFile (const std::string& filePath);
00196     bool saveToAiffFile (const std::string& filePath);
00197
00198     //=============================================================
00199     void clearAudioBuffer();
00200
00201     //=============================================================
00202     static inline AudioFileFormat determineAudioFileFormat (const std::vector<uint8_t>& fileData);
00203
00204     static inline int32_t fourBytesToInt (const std::vector<uint8_t>& source, int startIndex,
      Endianness endianness = Endianness::LittleEndian);
00205     static inline int16_t twoBytesToInt (const std::vector<uint8_t>& source, int startIndex,
      Endianness endianness = Endianness::LittleEndian);
00206     static inline int getIndexOfString (const std::vector<uint8_t>& source, std::string s);
00207     static inline int getIndexOfChunk (const std::vector<uint8_t>& source, const std::string&
      chunkHeaderID, int startIndex, Endianness endianness = Endianness::LittleEndian);
00208
00209     //=============================================================
00210     static inline uint32_t getAiffSampleRate (const std::vector<uint8_t>& fileData, int
      sampleRateStartIndex);
00211     static inline void addSampleRateToAiffData (std::vector<uint8_t>& fileData, uint32_t sampleRate);
00212
00213     //=============================================================
00214     static inline void addStringToFileData (std::vector<uint8_t>& fileData, std::string s);
00215     static inline void addInt32ToFileData (std::vector<uint8_t>& fileData, int32_t i, Endianness
      endianness = Endianness::LittleEndian);
```

```
00216     static inline void addInt16ToFileData (std::vector<uint8_t>& fileData, int16_t i, Endianness
      endianness = Endianness::LittleEndian);
00217
00218     //=============================================================
00219     static inline bool writeDataToFile (const std::vector<uint8_t>& fileData, std::string filePath);
00220
00221     //=============================================================
00222     void reportError (const std::string& errorMessage);
00223
00224     //=============================================================
00225     AudioFileFormat audioFileFormat;
00226     uint32_t sampleRate;
00227     int bitDepth;
00228     bool logErrorsToConsole {true};
00229 };
00230
00231 //=============================================================
00232 template <typename T>
00233 struct AudioSampleConverter
00234 {
00235     //=============================================================
00237     static T signedByteToSample (int8_t sample);
00238
00240     static int8_t sampleToSignedByte (T sample);
00241
00242     //=============================================================
00244     static T unsignedByteToSample (uint8_t sample);
00245
00247     static uint8_t sampleToUnsignedByte (T sample);
00248
00249     //=============================================================
00251     static T sixteenBitIntToSample (int16_t sample);
00252
00254     static int16_t sampleToSixteenBitInt (T sample);
00255
00256     //=============================================================
00258     static T twentyFourBitIntToSample (int32_t sample);
00259
00261     static int32_t sampleToTwentyFourBitInt (T sample);
00262
00263     //=============================================================
00265     static T thirtyTwoBitIntToSample (int32_t sample);
00266
00268     static int32_t sampleToThirtyTwoBitInt (T sample);
00269
00270     //=============================================================
00272     static T clamp (T v1, T minValue, T maxValue);
00273 };
00274
00275 //=============================================================
00276 struct AiffUtilities
00277 {
00278     //=============================================================
00280     static inline double decodeAiffSampleRate (const uint8_t* bytes);
00281
00283     static inline void encodeAiffSampleRate (double sampleRate, uint8_t* bytes);
00284 };
00285
00286 //=============================================================
00287 enum WavAudioFormat
00288 {
00289     PCM = 0x0001,
00290     IEEEFloat = 0x0003,
00291     ALaw = 0x0006,
00292     MULaw = 0x0007,
00293     Extensible = 0xFFFE
00294 };
00295
00296 //=============================================================
00297 enum AIFFAudioFormat
00298 {
00299     Uncompressed,
00300     Compressed,
00301     Error
00302 };
00303
00304 //=============================================================
00305 /* IMPLEMENTATION */
00306 //=============================================================
00307
00308 //=============================================================
00309 template <class T>
00310 AudioFile<T>::AudioFile()
00311 {
00312     bitDepth = 16;
00313     sampleRate = 44100;
00314     samples.resize (1);
```

```
00315     samples[0].resize (0);
00316     audioFileFormat = AudioFileFormat::NotLoaded;
00317 }
00318
00319 //=============================================================
00320 template <class T>
00321 AudioFile<T>::AudioFile (const std::string& filePath)
00322  :  AudioFile<T>()
00323 {
00324     load (filePath);
00325 }
00326
00327 //=============================================================
00328 template <class T>
00329 uint32_t AudioFile<T>::getSampleRate() const
00330 {
00331     return sampleRate;
00332 }
00333
00334 //=============================================================
00335 template <class T>
00336 int AudioFile<T>::getNumChannels() const
00337 {
00338     return (int)samples.size();
00339 }
00340
00341 //=============================================================
00342 template <class T>
00343 bool AudioFile<T>::isMono() const
00344 {
00345     return getNumChannels() == 1;
00346 }
00347
00348 //=============================================================
00349 template <class T>
00350 bool AudioFile<T>::isStereo() const
00351 {
00352     return getNumChannels() == 2;
00353 }
00354
00355 //=============================================================
00356 template <class T>
00357 int AudioFile<T>::getBitDepth() const
00358 {
00359     return bitDepth;
00360 }
00361
00362 //=============================================================
00363 template <class T>
00364 int AudioFile<T>::getNumSamplesPerChannel() const
00365 {
00366     if (samples.size() > 0)
00367         return (int) samples[0].size();
00368     else
00369         return 0;
00370 }
00371
00372 //=============================================================
00373 template <class T>
00374 double AudioFile<T>::getLengthInSeconds() const
00375 {
00376     return (double)getNumSamplesPerChannel() / (double)sampleRate;
00377 }
00378
00379 //=============================================================
00380 template <class T>
00381 void AudioFile<T>::printSummary() const
00382 {
00383     std::cerr « "|====================================|" « std::endl;
00384     std::cerr « "Num Channels: " « getNumChannels() « std::endl;
00385     std::cerr « "Num Samples Per Channel: " « getNumSamplesPerChannel() « std::endl;
00386     std::cerr « "Sample Rate: " « sampleRate « std::endl;
00387     std::cerr « "Bit Depth: " « bitDepth « std::endl;
00388     std::cerr « "Length in Seconds: " « getLengthInSeconds() « std::endl;
00389     std::cerr « "|====================================|" « std::endl;
00390 }
00391
00392 //=============================================================
00393 template <class T>
00394 bool AudioFile<T>::setAudioBuffer (const AudioBuffer& newBuffer)
00395 {
00396     int numChannels = (int)newBuffer.size();
00397
00398     if (numChannels <= 0)
00399     {
00400         assert (false && "The buffer you are trying to use has no channels");
00401         return false;
```

```
00402     }
00403
00404     size_t numSamples = newBuffer[0].size();
00405
00406     // set the number of channels
00407     samples.resize (newBuffer.size());
00408
00409     for (int k = 0; k < getNumChannels(); k++)
00410     {
00411         assert (newBuffer[k].size() == numSamples);
00412
00413         samples[k].resize (numSamples);
00414
00415         for (size_t i = 0; i < numSamples; i++)
00416         {
00417             samples[k][i] = newBuffer[k][i];
00418         }
00419     }
00420
00421     return true;
00422 }
00423
00424 //=============================================================
00425 template <class T>
00426 void AudioFile<T>::setAudioBufferSize (int numChannels, int numSamples)
00427 {
00428     samples.resize (numChannels);
00429     setNumSamplesPerChannel (numSamples);
00430 }
00431
00432 //=============================================================
00433 template <class T>
00434 void AudioFile<T>::setNumSamplesPerChannel (int numSamples)
00435 {
00436     int originalSize = getNumSamplesPerChannel();
00437
00438     for (int i = 0; i < getNumChannels();i++)
00439     {
00440         samples[i].resize (numSamples);
00441
00442         // set any new samples to zero
00443         if (numSamples > originalSize)
00444             std::fill (samples[i].begin() + originalSize, samples[i].end(), (T)0.);
00445     }
00446 }
00447
00448 //=============================================================
00449 template <class T>
00450 void AudioFile<T>::setNumChannels (int numChannels)
00451 {
00452     int originalNumChannels = getNumChannels();
00453     int originalNumSamplesPerChannel = getNumSamplesPerChannel();
00454
00455     samples.resize (numChannels);
00456
00457     // make sure any new channels are set to the right size
00458     // and filled with zeros
00459     if (numChannels > originalNumChannels)
00460     {
00461         for (int i = originalNumChannels; i < numChannels; i++)
00462         {
00463             samples[i].resize (originalNumSamplesPerChannel);
00464             std::fill (samples[i].begin(), samples[i].end(), (T)0.);
00465         }
00466     }
00467 }
00468
00469 //=============================================================
00470 template <class T>
00471 void AudioFile<T>::setBitDepth (int numBitsPerSample)
00472 {
00473     bitDepth = numBitsPerSample;
00474 }
00475
00476 //=============================================================
00477 template <class T>
00478 void AudioFile<T>::setSampleRate (uint32_t newSampleRate)
00479 {
00480     sampleRate = newSampleRate;
00481 }
00482
00483 //=============================================================
00484 template <class T>
00485 void AudioFile<T>::shouldLogErrorsToConsole (bool logErrors)
00486 {
00487     logErrorsToConsole = logErrors;
00488 }
```

```
00489
00490 //=============================================================
00491 template <class T>
00492 bool AudioFile<T>::load (const std::string& filePath)
00493 {
00494     std::ifstream file (filePath, std::ios::binary);
00495
00496     // check the file exists
00497     if (! file.good())
00498     {
00499         reportError ("ERROR: File doesn't exist or otherwise can't load file\n"  + filePath);
00500         return false;
00501     }
00502
00503     std::vector<uint8_t> fileData;
00504
00505     file.unsetf (std::ios::skipws);
00506
00507     file.seekg (0, std::ios::end);
00508     size_t length = file.tellg();
00509     file.seekg (0, std::ios::beg);
00510
00511     // allocate
00512     fileData.resize (length);
00513
00514     file.read(reinterpret_cast<char*> (fileData.data()), length);
00515     file.close();
00516
00517     if (file.gcount() != length)
00518     {
00519         reportError ("ERROR: Couldn't read entire file\n" + filePath);
00520         return false;
00521     }
00522
00523     // Handle very small files that will break our attempt to read the
00524     // first header info from them
00525     if (fileData.size() < 12)
00526     {
00527         reportError ("ERROR: File is not a valid audio file\n" + filePath);
00528         return false;
00529     }
00530     else
00531     {
00532         return loadFromMemory (fileData);
00533     }
00534 }
00535
00536 //=============================================================
00537 template <class T>
00538 bool AudioFile<T>::loadFromMemory (const std::vector<uint8_t>& fileData)
00539 {
00540     // get audio file format
00541     audioFileFormat = determineAudioFileFormat (fileData);
00542
00543     if (audioFileFormat == AudioFileFormat::Wave)
00544     {
00545         return decodeWaveFile (fileData);
00546     }
00547     else if (audioFileFormat == AudioFileFormat::Aiff)
00548     {
00549         return decodeAiffFile (fileData);
00550     }
00551     else
00552     {
00553         reportError ("Audio File Type: Error");
00554         return false;
00555     }
00556 }
00557
00558 //=============================================================
00559 template <class T>
00560 bool AudioFile<T>::decodeWaveFile (const std::vector<uint8_t>& fileData)
00561 {
00562     // -------------------------------------------------------------
00563     // HEADER CHUNK
00564     std::string headerChunkID (fileData.begin(), fileData.begin() + 4);
00565     //int32_t fileSizeInBytes = fourBytesToInt (fileData, 4) + 8;
00566     std::string format (fileData.begin() + 8, fileData.begin() + 12);
00567
00568     // -------------------------------------------------------------
00569     // try and find the start points of key chunks
00570     int indexOfDataChunk = getIndexOfChunk (fileData, "data", 12);
00571     int indexOfFormatChunk = getIndexOfChunk (fileData, "fmt ", 12);
00572     int indexOfXMLChunk = getIndexOfChunk (fileData, "iXML", 12);
00573
00574     // if we can't find the data or format chunks, or the IDs/formats don't seem to be as expected
00575     // then it is unlikely we'll able to read this file, so abort
```

```
00576     if (indexOfDataChunk == -1 || indexOfFormatChunk == -1 || headerChunkID != "RIFF" || format != "WAVE")
00577     {
00578         reportError ("ERROR: this doesn't seem to be a valid .WAV file");
00579         return false;
00580     }
00581
00582     // -----------------------------------------------------------
00583     // FORMAT CHUNK
00584     int f = indexOfFormatChunk;
00585     std::string formatChunkID (fileData.begin() + f, fileData.begin() + f + 4);
00586     //int32_t formatChunkSize = fourBytesToInt (fileData, f + 4);
00587     uint16_t audioFormat = twoBytesToInt (fileData, f + 8);
00588     uint16_t numChannels = twoBytesToInt (fileData, f + 10);
00589     sampleRate = (uint32_t) fourBytesToInt (fileData, f + 12);
00590     uint32_t numBytesPerSecond = fourBytesToInt (fileData, f + 16);
00591     uint16_t numBytesPerBlock = twoBytesToInt (fileData, f + 20);
00592     bitDepth = (int) twoBytesToInt (fileData, f + 22);
00593
00594     if (bitDepth > sizeof (T) * 8)
00595     {
00596         std::string message = "ERROR: you are trying to read a ";
00597         message += std::to_string (bitDepth);
00598         message += "-bit file using a ";
00599         message += std::to_string (sizeof (T) * 8);
00600         message += "-bit sample type";
00601         reportError (message);
00602         return false;
00603     }
00604
00605     uint16_t numBytesPerSample = static_cast<uint16_t> (bitDepth) / 8;
00606
00607     // check that the audio format is PCM or Float or extensible
00608     if (audioFormat != WavAudioFormat::PCM && audioFormat != WavAudioFormat::IEEEFloat && audioFormat != WavAudioFormat::Extensible)
00609     {
00610         reportError ("ERROR: this .WAV file is encoded in a format that this library does not support at present");
00611         return false;
00612     }
00613
00614     // check the number of channels is mono or stereo
00615     if (numChannels < 1 || numChannels > 128)
00616     {
00617         reportError ("ERROR: this WAV file seems to be an invalid number of channels (or corrupted?)");
00618         return false;
00619     }
00620
00621     // check header data is consistent
00622     if (numBytesPerSecond != static_cast<uint32_t> ((numChannels * sampleRate * bitDepth) / 8) || numBytesPerBlock != (numChannels * numBytesPerSample))
00623     {
00624         reportError ("ERROR: the header data in this WAV file seems to be inconsistent");
00625         return false;
00626     }
00627
00628     // check bit depth is either 8, 16, 24 or 32 bit
00629     if (bitDepth != 8 && bitDepth != 16 && bitDepth != 24 && bitDepth != 32)
00630     {
00631         reportError ("ERROR: this file has a bit depth that is not 8, 16, 24 or 32 bits");
00632         return false;
00633     }
00634
00635     // -----------------------------------------------------------
00636     // DATA CHUNK
00637     int d = indexOfDataChunk;
00638     std::string dataChunkID (fileData.begin() + d, fileData.begin() + d + 4);
00639     int32_t dataChunkSize = fourBytesToInt (fileData, d + 4);
00640
00641     int numSamples = dataChunkSize / (numChannels * bitDepth / 8);
00642     int samplesStartIndex = indexOfDataChunk + 8;
00643
00644     clearAudioBuffer();
00645     samples.resize (numChannels);
00646
00647     for (int i = 0; i < numSamples; i++)
00648     {
00649         for (int channel = 0; channel < numChannels; channel++)
00650         {
00651             int sampleIndex = samplesStartIndex + (numBytesPerBlock * i) + channel * numBytesPerSample;
00652
00653             if ((sampleIndex + (bitDepth / 8) - 1) >= fileData.size())
00654             {
00655                 reportError ("ERROR: read file error as the metadata indicates more samples than there are in the file data");
```

```
00656                 return false;
00657             }
00658
00659             if (bitDepth == 8)
00660             {
00661                 T sample = AudioSampleConverter<T>::unsignedByteToSample (fileData[sampleIndex]);
00662                 samples[channel].push_back (sample);
00663             }
00664             else if (bitDepth == 16)
00665             {
00666                 int16_t sampleAsInt = twoBytesToInt (fileData, sampleIndex);
00667                 T sample = AudioSampleConverter<T>::sixteenBitIntToSample (sampleAsInt);
00668                 samples[channel].push_back (sample);
00669             }
00670             else if (bitDepth == 24)
00671             {
00672                 int32_t sampleAsInt = 0;
00673                 sampleAsInt = (fileData[sampleIndex + 2] << 16) | (fileData[sampleIndex + 1] << 8) |
      fileData[sampleIndex];
00674
00675                 if (sampleAsInt & 0x800000) //  if the 24th bit is set, this is a negative number in
      24-bit world
00676                     sampleAsInt = sampleAsInt | ~0xFFFFFF; // so make sure sign is extended to the 32
      bit float
00677
00678                 T sample = AudioSampleConverter<T>::twentyFourBitIntToSample (sampleAsInt);
00679                 samples[channel].push_back (sample);
00680             }
00681             else if (bitDepth == 32)
00682             {
00683                 int32_t sampleAsInt = fourBytesToInt (fileData, sampleIndex);
00684                 T sample;
00685
00686                 if (audioFormat == WavAudioFormat::IEEEFloat && std::is_floating_point_v<T>)
00687                 {
00688                     float f;
00689                     memcpy (&f, &sampleAsInt, sizeof(int32_t));
00690                     sample = (T)f;
00691                 }
00692                 else // assume PCM
00693                 {
00694                     sample = AudioSampleConverter<T>::thirtyTwoBitIntToSample (sampleAsInt);
00695                 }
00696
00697                 samples[channel].push_back (sample);
00698             }
00699             else
00700             {
00701                 assert (false);
00702             }
00703         }
00704     }
00705
00706     // ------------------------------------------------------------
00707     // iXML CHUNK
00708     if (indexOfXMLChunk != -1)
00709     {
00710         int32_t chunkSize = fourBytesToInt (fileData, indexOfXMLChunk + 4);
00711         iXMLChunk = std::string ((const char*) &fileData[indexOfXMLChunk + 8], chunkSize);
00712     }
00713
00714     return true;
00715 }
00716
00717 //=============================================================
00718 template <class T>
00719 bool AudioFile<T>::decodeAiffFile (const std::vector<uint8_t>& fileData)
00720 {
00721     // -----------------------------------------------------------
00722     // HEADER CHUNK
00723     std::string headerChunkID (fileData.begin(), fileData.begin() + 4);
00724     //int32_t fileSizeInBytes = fourBytesToInt (fileData, 4, Endianness::BigEndian) + 8;
00725     std::string format (fileData.begin() + 8, fileData.begin() + 12);
00726
00727     int audioFormat = format == "AIFF" ? AIFFAudioFormat::Uncompressed : format == "AIFC" ?
      AIFFAudioFormat::Compressed : AIFFAudioFormat::Error;
00728
00729     // -----------------------------------------------------------
00730     // try and find the start points of key chunks
00731     int indexOfCommChunk = getIndexOfChunk (fileData, "COMM", 12, Endianness::BigEndian);
00732     int indexOfSoundDataChunk = getIndexOfChunk (fileData, "SSND", 12, Endianness::BigEndian);
00733     int indexOfXMLChunk = getIndexOfChunk (fileData, "iXML", 12, Endianness::BigEndian);
00734
00735     // if we can't find the data or format chunks, or the IDs/formats don't seem to be as expected
00736     // then it is unlikely we'll able to read this file, so abort
00737     if (indexOfSoundDataChunk == -1 || indexOfCommChunk == -1 || headerChunkID != "FORM" ||
      audioFormat == AIFFAudioFormat::Error)
```

```
00738     {
00739         reportError ("ERROR: this doesn't seem to be a valid AIFF file");
00740         return false;
00741     }
00742 
00743     // -----------------------------------------------------------
00744     // COMM CHUNK
00745     int p = indexOfCommChunk;
00746     std::string commChunkID (fileData.begin() + p, fileData.begin() + p + 4);
00747     //int32_t commChunkSize = fourBytesToInt (fileData, p + 4, Endianness::BigEndian);
00748     int16_t numChannels = twoBytesToInt (fileData, p + 8, Endianness::BigEndian);
00749     int32_t numSamplesPerChannel = fourBytesToInt (fileData, p + 10, Endianness::BigEndian);
00750     bitDepth = (int) twoBytesToInt (fileData, p + 14, Endianness::BigEndian);
00751     sampleRate = getAiffSampleRate (fileData, p + 16);
00752 
00753     if (bitDepth > sizeof (T) * 8)
00754     {
00755         std::string message = "ERROR: you are trying to read a ";
00756         message += std::to_string (bitDepth);
00757         message += "-bit file using a ";
00758         message += std::to_string (sizeof (T) * 8);
00759         message += "-bit sample type";
00760         reportError (message);
00761         return false;
00762     }
00763 
00764     // check the sample rate was properly decoded
00765     if (sampleRate == 0)
00766     {
00767         reportError ("ERROR: this AIFF file has an unsupported sample rate");
00768         return false;
00769     }
00770 
00771     // check the number of channels is mono or stereo
00772     if (numChannels < 1 ||numChannels > 2)
00773     {
00774         reportError ("ERROR: this AIFF file seems to be neither mono nor stereo (perhaps multi-track,
      or corrupted?)");
00775         return false;
00776     }
00777 
00778     // check bit depth is either 8, 16, 24 or 32-bit
00779     if (bitDepth != 8 && bitDepth != 16 && bitDepth != 24 && bitDepth != 32)
00780     {
00781         reportError ("ERROR: this file has a bit depth that is not 8, 16, 24 or 32 bits");
00782         return false;
00783     }
00784 
00785     // -----------------------------------------------------------
00786     // SSND CHUNK
00787     int s = indexOfSoundDataChunk;
00788     std::string soundDataChunkID (fileData.begin() + s, fileData.begin() + s + 4);
00789     int32_t soundDataChunkSize = fourBytesToInt (fileData, s + 4, Endianness::BigEndian);
00790     int32_t offset = fourBytesToInt (fileData, s + 8, Endianness::BigEndian);
00791     //int32_t blockSize = fourBytesToInt (fileData, s + 12, Endianness::BigEndian);
00792 
00793     int numBytesPerSample = bitDepth / 8;
00794     int numBytesPerFrame = numBytesPerSample * numChannels;
00795     int totalNumAudioSampleBytes = numSamplesPerChannel * numBytesPerFrame;
00796     int samplesStartIndex = s + 16 + (int)offset;
00797 
00798     // sanity check the data
00799     if ((soundDataChunkSize - 8) != totalNumAudioSampleBytes || totalNumAudioSampleBytes >
      static_cast<long>(fileData.size() - samplesStartIndex))
00800     {
00801         reportError ("ERROR: the metadatafor this file doesn't seem right");
00802         return false;
00803     }
00804 
00805     clearAudioBuffer();
00806     samples.resize (numChannels);
00807 
00808     for (int i = 0; i < numSamplesPerChannel; i++)
00809     {
00810         for (int channel = 0; channel < numChannels; channel++)
00811         {
00812             int sampleIndex = samplesStartIndex + (numBytesPerFrame * i) + channel *
      numBytesPerSample;
00813 
00814             if ((sampleIndex + (bitDepth / 8) - 1) >= fileData.size())
00815             {
00816                 reportError ("ERROR: read file error as the metadata indicates more samples than there
      are in the file data");
00817                 return false;
00818             }
00819 
00820             if (bitDepth == 8)
```

```
00821             {
00822                 T sample = AudioSampleConverter<T>::signedByteToSample (static_cast<int8_t>
      (fileData[sampleIndex]));
00823                 samples[channel].push_back (sample);
00824             }
00825             else if (bitDepth == 16)
00826             {
00827                 int16_t sampleAsInt = twoBytesToInt (fileData, sampleIndex, Endianness::BigEndian);
00828                 T sample = AudioSampleConverter<T>::sixteenBitIntToSample (sampleAsInt);
00829                 samples[channel].push_back (sample);
00830             }
00831             else if (bitDepth == 24)
00832             {
00833                 int32_t sampleAsInt = 0;
00834                 sampleAsInt = (fileData[sampleIndex] << 16) | (fileData[sampleIndex + 1] << 8) |
      fileData[sampleIndex + 2];
00835
00836                 if (sampleAsInt & 0x800000) //  if the 24th bit is set, this is a negative number in
      24-bit world
00837                     sampleAsInt = sampleAsInt | ~0xFFFFFF; // so make sure sign is extended to the 32
      bit float
00838
00839                 T sample = AudioSampleConverter<T>::twentyFourBitIntToSample (sampleAsInt);
00840                 samples[channel].push_back (sample);
00841             }
00842             else if (bitDepth == 32)
00843             {
00844                 int32_t sampleAsInt = fourBytesToInt (fileData, sampleIndex, Endianness::BigEndian);
00845                 T sample;
00846
00847                 if (audioFormat == AIFFAudioFormat::Compressed)
00848                     sample = (T)reinterpret_cast<float&> (sampleAsInt);
00849                 else // assume PCM
00850                     sample = AudioSampleConverter<T>::thirtyTwoBitIntToSample (sampleAsInt);
00851
00852                 samples[channel].push_back (sample);
00853             }
00854             else
00855             {
00856                 assert (false);
00857             }
00858         }
00859     }
00860
00861     // ------------------------------------------------------------
00862     // iXML CHUNK
00863     if (indexOfXMLChunk != -1)
00864     {
00865         int32_t chunkSize = fourBytesToInt (fileData, indexOfXMLChunk + 4);
00866         iXMLChunk = std::string ((const char*) &fileData[indexOfXMLChunk + 8], chunkSize);
00867     }
00868
00869     return true;
00870 }
00871
00872 //=============================================================
00873 template <class T>
00874 uint32_t AudioFile<T>::getAiffSampleRate (const std::vector<uint8_t>& fileData, int
      sampleRateStartIndex)
00875 {
00876     double sampleRate = AiffUtilities::decodeAiffSampleRate (&fileData[sampleRateStartIndex]);
00877     return static_cast<uint32_t> (sampleRate);
00878 }
00879
00880 //=============================================================
00881 template <class T>
00882 void AudioFile<T>::addSampleRateToAiffData (std::vector<uint8_t>& fileData, uint32_t sampleRateToAdd)
00883 {
00884     std::array<uint8_t, 10> sampleRateData;
00885     AiffUtilities::encodeAiffSampleRate (static_cast<double> (sampleRateToAdd),
      sampleRateData.data());
00886     fileData.insert (fileData.end(), sampleRateData.begin(), sampleRateData.end());
00887 }
00888
00889 //=============================================================
00890 template <class T>
00891 bool AudioFile<T>::save (const std::string& filePath, AudioFileFormat format)
00892 {
00893     if (format == AudioFileFormat::Wave)
00894     {
00895         return saveToWaveFile (filePath);
00896     }
00897     else if (format == AudioFileFormat::Aiff)
00898     {
00899         return saveToAiffFile (filePath);
00900     }
00901
```

```
00902     return false;
00903 }
00904
00905 //=============================================================
00906 template <class T>
00907 bool AudioFile<T>::saveToWaveFile (const std::string& filePath)
00908 {
00909     std::vector<uint8_t> fileData;
00910
00911     int32_t dataChunkSize = getNumSamplesPerChannel() * (getNumChannels() * bitDepth / 8);
00912     int16_t audioFormat = bitDepth == 32 && std::is_floating_point_v<T> ? WavAudioFormat::IEEEFloat :
      WavAudioFormat::PCM;
00913     int32_t formatChunkSize = audioFormat == WavAudioFormat::PCM ? 16 : 18;
00914     int32_t iXMLChunkSize = static_cast<int32_t> (iXMLChunk.size());
00915
00916     // -------------------------------------------------------------
00917     // HEADER CHUNK
00918     addStringToFileData (fileData, "RIFF");
00919
00920     // The file size in bytes is the header chunk size (4, not counting RIFF and WAVE) + the format
00921     // chunk size (24) + the metadata part of the data chunk plus the actual data chunk size
00922     int32_t fileSizeInBytes = 4 + formatChunkSize + 8 + 8 + dataChunkSize;
00923     if (iXMLChunkSize > 0)
00924     {
00925         fileSizeInBytes += (8 + iXMLChunkSize);
00926     }
00927
00928     addInt32ToFileData (fileData, fileSizeInBytes);
00929
00930     addStringToFileData (fileData, "WAVE");
00931
00932     // -------------------------------------------------------------
00933     // FORMAT CHUNK
00934     addStringToFileData (fileData, "fmt ");
00935     addInt32ToFileData (fileData, formatChunkSize); // format chunk size (16 for PCM)
00936     addInt16ToFileData (fileData, audioFormat); // audio format
00937     addInt16ToFileData (fileData, (int16_t)getNumChannels()); // num channels
00938     addInt32ToFileData (fileData, (int32_t)sampleRate); // sample rate
00939
00940     int32_t numBytesPerSecond = (int32_t) ((getNumChannels() * sampleRate * bitDepth) / 8);
00941     addInt32ToFileData (fileData, numBytesPerSecond);
00942
00943     int16_t numBytesPerBlock = getNumChannels() * (bitDepth / 8);
00944     addInt16ToFileData (fileData, numBytesPerBlock);
00945
00946     addInt16ToFileData (fileData, (int16_t)bitDepth);
00947
00948     if (audioFormat == WavAudioFormat::IEEEFloat)
00949         addInt16ToFileData (fileData, 0); // extension size
00950
00951     // -------------------------------------------------------------
00952     // DATA CHUNK
00953     addStringToFileData (fileData, "data");
00954     addInt32ToFileData (fileData, dataChunkSize);
00955
00956     for (int i = 0; i < getNumSamplesPerChannel(); i++)
00957     {
00958         for (int channel = 0; channel < getNumChannels(); channel++)
00959         {
00960             if (bitDepth == 8)
00961             {
00962                 uint8_t byte = AudioSampleConverter<T>::sampleToUnsignedByte (samples[channel][i]);
00963                 fileData.push_back (byte);
00964             }
00965             else if (bitDepth == 16)
00966             {
00967                 int16_t sampleAsInt = AudioSampleConverter<T>::sampleToSixteenBitInt
      (samples[channel][i]);
00968                 addInt16ToFileData (fileData, sampleAsInt);
00969             }
00970             else if (bitDepth == 24)
00971             {
00972                 int32_t sampleAsIntAgain = AudioSampleConverter<T>::sampleToTwentyFourBitInt
      (samples[channel][i]);
00973
00974                 uint8_t bytes[3];
00975                 bytes[2] = (uint8_t) (sampleAsIntAgain >> 16) & 0xFF;
00976                 bytes[1] = (uint8_t) (sampleAsIntAgain >>  8) & 0xFF;
00977                 bytes[0] = (uint8_t) sampleAsIntAgain & 0xFF;
00978
00979                 fileData.push_back (bytes[0]);
00980                 fileData.push_back (bytes[1]);
00981                 fileData.push_back (bytes[2]);
00982             }
00983             else if (bitDepth == 32)
00984             {
00985                 int32_t sampleAsInt;
```

```
00986
00987                 if (audioFormat == WavAudioFormat::IEEEFloat)
00988                     sampleAsInt = (int32_t) reinterpret_cast<int32_t&> (samples[channel][i]);
00989                 else // assume PCM
00990                     sampleAsInt = AudioSampleConverter<T>::sampleToThirtyTwoBitInt
      (samples[channel][i]);
00991
00992                 addInt32ToFileData (fileData, sampleAsInt, Endianness::LittleEndian);
00993             }
00994             else
00995             {
00996                 assert (false && "Trying to write a file with unsupported bit depth");
00997                 return false;
00998             }
00999         }
01000     }
01001
01002     // -----------------------------------------------------------
01003     // iXML CHUNK
01004     if (iXMLChunkSize > 0)
01005     {
01006         addStringToFileData (fileData, "iXML");
01007         addInt32ToFileData (fileData, iXMLChunkSize);
01008         addStringToFileData (fileData, iXMLChunk);
01009     }
01010
01011     // check that the various sizes we put in the metadata are correct
01012     if (fileSizeInBytes != static_cast<int32_t> (fileData.size() - 8) || dataChunkSize !=
      (getNumSamplesPerChannel() * getNumChannels() * (bitDepth / 8)))
01013     {
01014         reportError ("ERROR: couldn't save file to " + filePath);
01015         return false;
01016     }
01017
01018     // try to write the file
01019     return writeDataToFile (fileData, filePath);
01020 }
01021
01022 //=============================================================
01023 template <class T>
01024 bool AudioFile<T>::saveToAiffFile (const std::string& filePath)
01025 {
01026     std::vector<uint8_t> fileData;
01027
01028     int32_t numBytesPerSample = bitDepth / 8;
01029     int32_t numBytesPerFrame = numBytesPerSample * getNumChannels();
01030     int32_t totalNumAudioSampleBytes = getNumSamplesPerChannel() * numBytesPerFrame;
01031     int32_t soundDataChunkSize = totalNumAudioSampleBytes + 8;
01032     int32_t iXMLChunkSize = static_cast<int32_t> (iXMLChunk.size());
01033
01034     // -----------------------------------------------------------
01035     // HEADER CHUNK
01036     addStringToFileData (fileData, "FORM");
01037
01038     // The file size in bytes is the header chunk size (4, not counting FORM and AIFF) + the COMM
01039     // chunk size (26) + the metadata part of the SSND chunk plus the actual data chunk size
01040     int32_t fileSizeInBytes = 4 + 26 + 16 + totalNumAudioSampleBytes;
01041     if (iXMLChunkSize > 0)
01042     {
01043         fileSizeInBytes += (8 + iXMLChunkSize);
01044     }
01045
01046     addInt32ToFileData (fileData, fileSizeInBytes, Endianness::BigEndian);
01047
01048     addStringToFileData (fileData, "AIFF");
01049
01050     // -----------------------------------------------------------
01051     // COMM CHUNK
01052     addStringToFileData (fileData, "COMM");
01053     addInt32ToFileData (fileData, 18, Endianness::BigEndian); // commChunkSize
01054     addInt16ToFileData (fileData, getNumChannels(), Endianness::BigEndian); // num channels
01055     addInt32ToFileData (fileData, getNumSamplesPerChannel(), Endianness::BigEndian); // num samples
      per channel
01056     addInt16ToFileData (fileData, bitDepth, Endianness::BigEndian); // bit depth
01057     addSampleRateToAiffData (fileData, sampleRate);
01058
01059     // -----------------------------------------------------------
01060     // SSND CHUNK
01061     addStringToFileData (fileData, "SSND");
01062     addInt32ToFileData (fileData, soundDataChunkSize, Endianness::BigEndian);
01063     addInt32ToFileData (fileData, 0, Endianness::BigEndian); // offset
01064     addInt32ToFileData (fileData, 0, Endianness::BigEndian); // block size
01065
01066     for (int i = 0; i < getNumSamplesPerChannel(); i++)
01067     {
01068         for (int channel = 0; channel < getNumChannels(); channel++)
01069         {
```

```
01070             if (bitDepth == 8)
01071             {
01072                 uint8_t byte = static_cast<uint8_t> (AudioSampleConverter<T>::sampleToSignedByte
      (samples[channel][i]));
01073                 fileData.push_back (byte);
01074             }
01075             else if (bitDepth == 16)
01076             {
01077                 int16_t sampleAsInt = AudioSampleConverter<T>::sampleToSixteenBitInt
      (samples[channel][i]);
01078                 addInt16ToFileData (fileData, sampleAsInt, Endianness::BigEndian);
01079             }
01080             else if (bitDepth == 24)
01081             {
01082                 int32_t sampleAsIntAgain = AudioSampleConverter<T>::sampleToTwentyFourBitInt
      (samples[channel][i]);
01083
01084                 uint8_t bytes[3];
01085                 bytes[0] = (uint8_t) (sampleAsIntAgain » 16) & 0xFF;
01086                 bytes[1] = (uint8_t) (sampleAsIntAgain »  8) & 0xFF;
01087                 bytes[2] = (uint8_t) sampleAsIntAgain & 0xFF;
01088
01089                 fileData.push_back (bytes[0]);
01090                 fileData.push_back (bytes[1]);
01091                 fileData.push_back (bytes[2]);
01092             }
01093             else if (bitDepth == 32)
01094             {
01095                 // write samples as signed integers (no implementation yet for floating point, but
      looking at WAV implementation should help)
01096                 int32_t sampleAsInt = AudioSampleConverter<T>::sampleToThirtyTwoBitInt
      (samples[channel][i]);
01097                 addInt32ToFileData (fileData, sampleAsInt, Endianness::BigEndian);
01098             }
01099             else
01100             {
01101                 assert (false && "Trying to write a file with unsupported bit depth");
01102                 return false;
01103             }
01104         }
01105     }
01106
01107     // -----------------------------------------------------------
01108     // iXML CHUNK
01109     if (iXMLChunkSize > 0)
01110     {
01111         addStringToFileData (fileData, "iXML");
01112         addInt32ToFileData (fileData, iXMLChunkSize, Endianness::BigEndian);
01113         addStringToFileData (fileData, iXMLChunk);
01114     }
01115
01116     // check that the various sizes we put in the metadata are correct
01117     if (fileSizeInBytes != static_cast<int32_t> (fileData.size() - 8) || soundDataChunkSize !=
      getNumSamplesPerChannel() *  numBytesPerFrame + 8)
01118     {
01119         reportError ("ERROR: couldn't save file to " + filePath);
01120         return false;
01121     }
01122
01123     // try to write the file
01124     return writeDataToFile (fileData, filePath);
01125 }
01126
01127 //=============================================================
01128 template <class T>
01129 bool AudioFile<T>::writeDataToFile (const std::vector<uint8_t>& fileData, std::string filePath)
01130 {
01131     std::ofstream outputFile (filePath, std::ios::binary);
01132
01133     if (!outputFile.is_open())
01134     {
01135         return false;
01136     }
01137
01138     outputFile.write ((const char*)fileData.data(), fileData.size());
01139     outputFile.close();
01140     return true;
01141 }
01142
01143 //=============================================================
01144 template <class T>
01145 void AudioFile<T>::addStringToFileData (std::vector<uint8_t>& fileData, std::string s)
01146 {
01147     for (size_t i = 0; i < s.length();i++)
01148         fileData.push_back ((uint8_t) s[i]);
01149 }
01150
```

```
01151 //=============================================================
01152 template <class T>
01153 void AudioFile<T>::addInt32ToFileData (std::vector<uint8_t>& fileData, int32_t i, Endianness
      endianness)
01154 {
01155     uint8_t bytes[4];
01156 
01157     if (endianness == Endianness::LittleEndian)
01158     {
01159         bytes[3] = (i » 24) & 0xFF;
01160         bytes[2] = (i » 16) & 0xFF;
01161         bytes[1] = (i » 8) & 0xFF;
01162         bytes[0] = i & 0xFF;
01163     }
01164     else
01165     {
01166         bytes[0] = (i » 24) & 0xFF;
01167         bytes[1] = (i » 16) & 0xFF;
01168         bytes[2] = (i » 8) & 0xFF;
01169         bytes[3] = i & 0xFF;
01170     }
01171 
01172     for (int j = 0; j < 4; j++)
01173         fileData.push_back (bytes[j]);
01174 }
01175 
01176 //=============================================================
01177 template <class T>
01178 void AudioFile<T>::addInt16ToFileData (std::vector<uint8_t>& fileData, int16_t i, Endianness
      endianness)
01179 {
01180     uint8_t bytes[2];
01181 
01182     if (endianness == Endianness::LittleEndian)
01183     {
01184         bytes[1] = (i » 8) & 0xFF;
01185         bytes[0] = i & 0xFF;
01186     }
01187     else
01188     {
01189         bytes[0] = (i » 8) & 0xFF;
01190         bytes[1] = i & 0xFF;
01191     }
01192 
01193     fileData.push_back (bytes[0]);
01194     fileData.push_back (bytes[1]);
01195 }
01196 
01197 //=============================================================
01198 template <class T>
01199 void AudioFile<T>::clearAudioBuffer()
01200 {
01201     for (size_t i = 0; i < samples.size();i++)
01202     {
01203         samples[i].clear();
01204     }
01205 
01206     samples.clear();
01207 }
01208 
01209 //=============================================================
01210 template <class T>
01211 AudioFileFormat AudioFile<T>::determineAudioFileFormat (const std::vector<uint8_t>& fileData)
01212 {
01213     if (fileData.size() < 4)
01214         return AudioFileFormat::Error;
01215 
01216     std::string header (fileData.begin(), fileData.begin() + 4);
01217 
01218     if (header == "RIFF")
01219         return AudioFileFormat::Wave;
01220     else if (header == "FORM")
01221         return AudioFileFormat::Aiff;
01222     else
01223         return AudioFileFormat::Error;
01224 }
01225 
01226 //=============================================================
01227 template <class T>
01228 int32_t AudioFile<T>::fourBytesToInt (const std::vector<uint8_t>& source, int startIndex, Endianness
      endianness)
01229 {
01230     if (source.size() >= (startIndex + 4))
01231     {
01232         int32_t result;
01233 
01234         if (endianness == Endianness::LittleEndian)
```

```
01235             result = (source[startIndex + 3] << 24) | (source[startIndex + 2] << 16) |
      (source[startIndex + 1] << 8) | source[startIndex];
01236         else
01237             result = (source[startIndex] << 24) | (source[startIndex + 1] << 16) | (source[startIndex +
      2] << 8) | source[startIndex + 3];
01238
01239         return result;
01240     }
01241     else
01242     {
01243         assert (false && "Attempted to read four bytes from vector at position where out of bounds
      access would occur");
01244         return 0; // this is a dummy value as we don't have one to return
01245     }
01246 }
01247
01248 //=============================================================
01249 template <class T>
01250 int16_t AudioFile<T>::twoBytesToInt (const std::vector<uint8_t>& source, int startIndex, Endianness
      endianness)
01251 {
01252     int16_t result;
01253
01254     if (endianness == Endianness::LittleEndian)
01255         result = (source[startIndex + 1] << 8) | source[startIndex];
01256     else
01257         result = (source[startIndex] << 8) | source[startIndex + 1];
01258
01259     return result;
01260 }
01261
01262 //=============================================================
01263 template <class T>
01264 int AudioFile<T>::getIndexOfChunk (const std::vector<uint8_t>& source, const std::string&
      chunkHeaderID, int startIndex, Endianness endianness)
01265 {
01266     constexpr int dataLen = 4;
01267
01268     if (chunkHeaderID.size() != dataLen)
01269     {
01270         assert (false && "Invalid chunk header ID string");
01271         return -1;
01272     }
01273
01274     int i = startIndex;
01275     while (i < source.size() - dataLen)
01276     {
01277         if (memcmp (&source[i], chunkHeaderID.data(), dataLen) == 0)
01278         {
01279             return i;
01280         }
01281
01282         i += dataLen;
01283
01284         // If somehow we don't have 4 bytes left to read, then exit with -1
01285         if ((i + 4) >= source.size())
01286             return -1;
01287
01288         int32_t chunkSize = fourBytesToInt (source, i, endianness);
01289         // Assume chunk size is invalid if it's greater than the number of bytes remaining in source
01290         if (chunkSize > (source.size() - i - dataLen) || (chunkSize < 0))
01291         {
01292             assert (false && "Invalid chunk size");
01293             return -1;
01294         }
01295         i += (dataLen + chunkSize);
01296     }
01297
01298     return -1;
01299 }
01300
01301 //=============================================================
01302 template <class T>
01303 void AudioFile<T>::reportError (const std::string& errorMessage)
01304 {
01305     if (logErrorsToConsole)
01306         std::cerr << errorMessage << std::endl;
01307 }
01308
01309 //=============================================================
01310 template <typename SignedType>
01311 typename std::make_unsigned<SignedType>::type convertSignedToUnsigned (SignedType signedValue)
01312 {
01313     static_assert (std::is_signed<SignedType>::value, "The input value must be signed");
01314
01315     typename std::make_unsigned<SignedType>::type unsignedValue = static_cast<typename
      std::make_unsigned<SignedType>::type> (1) + std::numeric_limits<SignedType>::max();
```

```
01316
01317     unsignedValue += signedValue;
01318     return unsignedValue;
01319 }
01320
01321 //============================================================
01322 enum SampleLimit
01323 {
01324     SignedInt16_Min = -32768,
01325     SignedInt16_Max = 32767,
01326     UnsignedInt16_Min = 0,
01327     UnsignedInt16_Max = 65535,
01328     SignedInt24_Min = -8388608,
01329     SignedInt24_Max = 8388607,
01330     UnsignedInt24_Min = 0,
01331     UnsignedInt24_Max = 16777215
01332 };
01333
01334 //============================================================
01335 template <class T>
01336 T AudioSampleConverter<T>::thirtyTwoBitIntToSample (int32_t sample)
01337 {
01338     if constexpr (std::is_floating_point<T>::value)
01339     {
01340         return static_cast<T> (sample) / static_cast<T> (std::numeric_limits<int32_t>::max());
01341     }
01342     else if (std::numeric_limits<T>::is_integer)
01343     {
01344         if constexpr (std::is_signed_v<T>)
01345             return static_cast<T> (sample);
01346         else
01347             return static_cast<T> (clamp (static_cast<T> (sample + 2147483648), 0, 4294967295));
01348     }
01349 }
01350
01351 //============================================================
01352 template <class T>
01353 int32_t AudioSampleConverter<T>::sampleToThirtyTwoBitInt (T sample)
01354 {
01355     if constexpr (std::is_floating_point<T>::value)
01356     {
01357         // multiplying a float by a the max int32_t is problematic because
01358         // of roundng errors which can cause wrong values to come out, so
01359         // we use a different implementation here compared to other types
01360         if constexpr (std::is_same_v<T, float>)
01361         {
01362             if (sample >= 1.f)
01363                 return std::numeric_limits<int32_t>::max();
01364             else if (sample <= -1.f)
01365                 return std::numeric_limits<int32_t>::lowest() + 1; // starting at 1 preserves symmetry
01366             else
01367                 return static_cast<int32_t> (sample * std::numeric_limits<int32_t>::max());
01368         }
01369         else
01370         {
01371             return static_cast<int32_t> (clamp (sample, -1., 1.) *
    std::numeric_limits<int32_t>::max());
01372         }
01373     }
01374     else
01375     {
01376         if constexpr (std::is_signed_v<T>)
01377             return static_cast<int32_t> (clamp (sample, -2147483648LL, 2147483647LL));
01378         else
01379             return static_cast<int32_t> (clamp (sample, 0, 4294967295) - 2147483648);
01380     }
01381 }
01382
01383 //============================================================
01384 template <class T>
01385 T AudioSampleConverter<T>::twentyFourBitIntToSample (int32_t sample)
01386 {
01387     if constexpr (std::is_floating_point<T>::value)
01388     {
01389         return static_cast<T> (sample) / static_cast<T> (8388607.);
01390     }
01391     else if (std::numeric_limits<T>::is_integer)
01392     {
01393         if constexpr (std::is_signed_v<T>)
01394             return static_cast<T> (clamp (sample, SignedInt24_Min, SignedInt24_Max));
01395         else
01396             return static_cast<T> (clamp (sample + 8388608, UnsignedInt24_Min, UnsignedInt24_Max));
01397     }
01398 }
01399
01400 //============================================================
01401 template <class T>
```

```
01402 int32_t AudioSampleConverter<T>::sampleToTwentyFourBitInt (T sample)
01403 {
01404     if constexpr (std::is_floating_point<T>::value)
01405     {
01406         sample = clamp (sample, -1., 1.);
01407         return static_cast<int32_t> (sample * 8388607.);
01408     }
01409     else
01410     {
01411         if constexpr (std::is_signed_v<T>)
01412             return static_cast<int32_t> (clamp (sample, SignedInt24_Min, SignedInt24_Max));
01413         else
01414             return static_cast<int32_t> (clamp (sample, UnsignedInt24_Min, UnsignedInt24_Max) +
      SignedInt24_Min);
01415     }
01416 }
01417
01418 //=============================================================
01419 template <class T>
01420 T AudioSampleConverter<T>::sixteenBitIntToSample (int16_t sample)
01421 {
01422     if constexpr (std::is_floating_point<T>::value)
01423     {
01424         return static_cast<T> (sample) / static_cast<T> (32767.);
01425     }
01426     else if constexpr (std::numeric_limits<T>::is_integer)
01427     {
01428         if constexpr (std::is_signed_v<T>)
01429             return static_cast<T> (sample);
01430         else
01431             return static_cast<T> (convertSignedToUnsigned<int16_t> (sample));
01432     }
01433 }
01434
01435 //=============================================================
01436 template <class T>
01437 int16_t AudioSampleConverter<T>::sampleToSixteenBitInt (T sample)
01438 {
01439     if constexpr (std::is_floating_point<T>::value)
01440     {
01441         sample = clamp (sample, -1., 1.);
01442         return static_cast<int16_t> (sample * 32767.);
01443     }
01444     else
01445     {
01446         if constexpr (std::is_signed_v<T>)
01447             return static_cast<int16_t> (clamp (sample, SignedInt16_Min, SignedInt16_Max));
01448         else
01449             return static_cast<int16_t> (clamp (sample, UnsignedInt16_Min, UnsignedInt16_Max) +
      SignedInt16_Min);
01450     }
01451 }
01452
01453 //=============================================================
01454 template <class T>
01455 uint8_t AudioSampleConverter<T>::sampleToUnsignedByte (T sample)
01456 {
01457     if constexpr (std::is_floating_point<T>::value)
01458     {
01459         sample = clamp (sample, -1., 1.);
01460         sample = (sample + 1.) / 2.;
01461         return static_cast<uint8_t> (1 + (sample * 254));
01462     }
01463     else
01464     {
01465         if constexpr (std::is_signed_v<T>)
01466             return static_cast<uint8_t> (clamp (sample, -128, 127) + 128);
01467         else
01468             return static_cast<uint8_t> (clamp (sample, 0, 255));
01469     }
01470 }
01471
01472 //=============================================================
01473 template <class T>
01474 int8_t AudioSampleConverter<T>::sampleToSignedByte (T sample)
01475 {
01476     if constexpr (std::is_floating_point<T>::value)
01477     {
01478         sample = clamp (sample, -1., 1.);
01479         return static_cast<int8_t> (sample * (T)0x7F);
01480     }
01481     else
01482     {
01483         if constexpr (std::is_signed_v<T>)
01484             return static_cast<int8_t> (clamp (sample, -128, 127));
01485         else
01486             return static_cast<int8_t> (clamp (sample, 0, 255) - 128);
```

```
01487     }
01488 }
01489
01490 //==========================================================
01491 template <class T>
01492 T AudioSampleConverter<T>::unsignedByteToSample (uint8_t sample)
01493 {
01494     if constexpr (std::is_floating_point<T>::value)
01495     {
01496         return static_cast<T> (sample - 128) / static_cast<T> (127.);
01497     }
01498     else if (std::numeric_limits<T>::is_integer)
01499     {
01500         if constexpr (std::is_unsigned_v<T>)
01501             return static_cast<T> (sample);
01502         else
01503             return static_cast<T> (sample - 128);
01504     }
01505 }
01506
01507 //==========================================================
01508 template <class T>
01509 T AudioSampleConverter<T>::signedByteToSample (int8_t sample)
01510 {
01511     if constexpr (std::is_floating_point<T>::value)
01512     {
01513         return static_cast<T> (sample) / static_cast<T> (127.);
01514     }
01515     else if constexpr (std::numeric_limits<T>::is_integer)
01516     {
01517         if constexpr (std::is_signed_v<T>)
01518             return static_cast<T> (sample);
01519         else
01520             return static_cast<T> (convertSignedToUnsigned<int8_t> (sample));
01521     }
01522 }
01523
01524 //==========================================================
01525 template <class T>
01526 T AudioSampleConverter<T>::clamp (T value, T minValue, T maxValue)
01527 {
01528     value = std::min (value, maxValue);
01529     value = std::max (value, minValue);
01530     return value;
01531 }
01532
01533 //==========================================================
01534 inline double AiffUtilities::decodeAiffSampleRate (const uint8_t* bytes)
01535 {
01536     // Note: Sample rate is 80 bits made up of
01537     // * 1 sign bit
01538     // * 15 exponent bits
01539     // * 64 mantissa bits
01540
01541     // ------------------------------------------------
01542     // Sign
01543
01544     // Extract the sign (most significant bit of byte 0)
01545     int sign = (bytes[0] & 0x80) ? -1 : 1;
01546
01547     // ------------------------------------------------
01548     // Exponent
01549
01550     // byte 0: ignore the sign and shift the most significant bits to the left by one byte
01551     uint16_t msbShifted = (static_cast<uint16_t> (bytes[0] & 0x7F) << 8);
01552
01553     // calculate exponent by combining byte 0 and byte 1 and subtract bias
01554     uint16_t exponent = (msbShifted | static_cast<uint16_t> (bytes[1])) - 16383;
01555
01556     // ------------------------------------------------
01557     // Mantissa
01558
01559     // Extract the mantissa (remaining 64 bits) by looping over the remaining
01560     // bytes and combining them while shifting the result to the left by
01561     // 8 bits each time
01562     uint64_t mantissa = 0;
01563
01564     for (int i = 2; i < 10; ++i)
01565         mantissa = (mantissa << 8) | bytes[i];
01566
01567     // Normalize the mantissa (implicit leading 1 for normalized values)
01568     double normalisedMantissa = static_cast<double> (mantissa) / (1ULL << 63);
01569
01570     // ------------------------------------------------
01571     // Combine sign, exponent, and mantissa into a double
01572
01573     return sign * std::ldexp (normalisedMantissa, exponent);
```

```
01574 }
01575
01576 //==============================================================
01577 inline void AiffUtilities::encodeAiffSampleRate (double sampleRate, uint8_t* bytes)
01578 {
01579     // Determine the sign
01580     int sign = (sampleRate < 0) ? -1 : 1;
01581
01582     if (sign == -1)
01583         sampleRate = -sampleRate;
01584
01585     // Set most significant bit of byte 0 for the sign
01586     bytes[0] = (sign == -1) ? 0x80 : 0x00;
01587
01588     // Calculate the exponent using logarithm (log base 2)
01589     int exponent = (log (sampleRate) / log (2.0));
01590
01591     // Add bias to exponent for AIFF
01592     uint16_t biasedExponent = static_cast<uint16_t> (exponent + 16383);
01593
01594     // Normalize the sample rate
01595     double normalizedSampleRate = sampleRate / pow (2.0, exponent);
01596
01597     // Calculate the mantissa
01598     uint64_t mantissa = static_cast<uint64_t> (normalizedSampleRate * (1ULL << 63));
01599
01600     // Pack the exponent into first two bytes of 10-byte AIFF format
01601     bytes[0] |= (biasedExponent >> 8) & 0x7F;    // Upper 7 bits of exponent
01602     bytes[1] = biasedExponent & 0xFF;            // Lower 8 bits of exponent
01603
01604     // Put the mantissa into byte array
01605     for (int i = 0; i < 8; ++i)
01606         bytes[2 + i] = (mantissa >> (8 * (7 - i))) & 0xFF;
01607 }
01608
01609 #if defined (_MSC_VER)
01610     __pragma(warning (pop))
01611 #elif defined (__GNUC__)
01612     _Pragma("GCC diagnostic pop")
01613 #endif
01614
01615 #endif /* AudioFile_h */
```

## 7.8 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)

## 7.9 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
00071         void GPIOStop();
00072
00073
00074         void SetCallback(GPIOCallback cb, void* context);
00075         static void IMUMathsCallback(void* context, float X, float Y, float Z){
00076             IMUMathsName::IMUMaths* maths = static_cast<IMUMathsName::IMUMaths*>(context);
00077             maths->SoundChecker(X,Y,Z);
00078         }
00079
00080     };
00081 }
00082
00083 #endif
```

## 7.10 src/libs/I2C/include/icm20948_defs.hpp File Reference

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

### Macros

- #define ICM20948_I2C_ADDR 0x69
- #define ICM20948_MAGN_I2C_ADDR 0x0C
- #define ICM20948_WHO_AM_I_BANK 0
- #define ICM20948_USER_CTRL_BANK 0
- #define ICM20948_LP_CONFIG_BANK 0
- #define ICM20948_PWR_MGMT_1_BANK 0
- #define ICM20948_PWR_MGMT_2_BANK 0
- #define ICM20948_INT_PIN_CFG_BANK 0
- #define ICM20948_I2C_MST_STATUS_BANK 0
- #define ICM20948_ACCEL_OUT_BANK 0
- #define ICM20948_GYRO_OUT_BANK 0
- #define ICM20948_TEMP_OUT_BANK 0
- #define ICM20948_EXT_SLV_SENS_DATA_00_BANK 0
- #define ICM20948_INT_ENABLE_BANK 0
- #define ICM20948_INT_ENABLE_1_BANK 0
- #define ICM20948_INT_ENABLE_2_BANK 0
- #define ICM20948_INT_ENABLE_3_BANK 0
- #define ICM20948_INT_STATUS_BANK 0
- #define ICM20948_INT_STATUS_1_BANK 0
- #define ICM20948_GYRO_SMPLRT_DIV_BANK 2
- #define ICM20948_GYRO_CONFIG_1_BANK 2
- #define ICM20948_ACCEL_SMPLRT_DIV_1_BANK 2
- #define ICM20948_ACCEL_SMPLRT_DIV_2_BANK 2
- #define ICM20948_ACCEL_CONFIG_1_BANK 2
- #define ICM20948_ACCEL_INTEL_CTRL_BANK 2
- #define ICM20948_ACCEL_WOM_THR_BANK 2
- #define ICM20948_I2C_MST_CTRL_BANK 3
- #define ICM20948_I2C_SLV0_ADDR_BANK 3
- #define ICM20948_I2C_SLV0_REG_BANK 3
- #define ICM20948_I2C_SLV0_CTRL_BANK 3
- #define ICM20948_I2C_SLV4_ADDR_BANK 3
- #define ICM20948_I2C_SLV4_REG_BANK 3
- #define ICM20948_I2C_SLV4_CTRL_BANK 3
- #define ICM20948_I2C_SLV4_DO_BANK 3
- #define ICM20948_I2C_SLV4_DI_BANK 3
- #define ICM20948_REG_BANK_SEL_ADDR 0x7F
- #define ICM20948_WHO_AM_I_ADDR 0x00
- #define ICM20948_USER_CTRL_ADDR 0x03
- #define ICM20948_LP_CONFIG_ADDR 0x05
- #define ICM20948_PWR_MGMT_1_ADDR 0x06
- #define ICM20948_PWR_MGMT_2_ADDR 0x07
- #define ICM20948_INT_PIN_CFG_ADDR 0x0F
- #define ICM20948_I2C_MST_STATUS_ADDR 0x17
- #define ICM20948_ACCEL_XOUT_H_ADDR 0x2D
- #define ICM20948_ACCEL_XOUT_L_ADDR 0x2E
- #define ICM20948_ACCEL_YOUT_H_ADDR 0x2F
- #define ICM20948_ACCEL_YOUT_L_ADDR 0x30
- #define ICM20948_ACCEL_ZOUT_H_ADDR 0x31
- #define ICM20948_ACCEL_ZOUT_L_ADDR 0x32
- #define ICM20948_GYRO_XOUT_H_ADDR 0x33
- #define ICM20948_GYRO_XOUT_L_ADDR 0x34
- #define ICM20948_GYRO_YOUT_H_ADDR 0x35
- #define ICM20948_GYRO_YOUT_L_ADDR 0x36

- #define ICM20948_GYRO_ZOUT_H_ADDR 0x37
- #define ICM20948_GYRO_ZOUT_L_ADDR 0x38
- #define ICM20948_TEMP_OUT_H_ADDR 0x39
- #define ICM20948_TEMP_OUT_L_ADDR 0x3A
- #define ICM20948_EXT_SLV_SENS_DATA_00_ADDR 0x3B
- #define ICM20948_INT_ENABLE_ADDR 0x10
- #define ICM20948_INT_ENABLE_1_ADDR 0x11
- #define ICM20948_INT_STATUS_ADDR 0x19
- #define ICM20948_INT_STATUS_1_ADDR 0x1A
- #define ICM20948_GYRO_SMPLRT_DIV_ADDR 0x00
- #define ICM20948_GYRO_CONFIG_1_ADDR 0x01
- #define ICM20948_ACCEL_SMPLRT_DIV_1_ADDR 0x10
- #define ICM20948_ACCEL_SMPLRT_DIV_2_ADDR 0x11
- #define ICM20948_ACCEL_CONFIG_1_ADDR 0x14
- #define ICM20948_ACCEL_INTEL_CTRL_ADDR 0x12
- #define ICM20948_ACCEL_WOM_THR_ADDR 0x13
- #define ICM20948_I2C_MST_CTRL_ADDR 0x01
- #define ICM20948_I2C_SLV0_ADDR_ADDR 0x03
- #define ICM20948_I2C_SLV0_REG_ADDR 0x04
- #define ICM20948_I2C_SLV0_CTRL_ADDR 0x05
- #define ICM20948_I2C_SLV4_ADDR_ADDR 0x13
- #define ICM20948_I2C_SLV4_REG_ADDR 0x14
- #define ICM20948_I2C_SLV4_CTRL_ADDR 0x15
- #define ICM20948_I2C_SLV4_DO_ADDR 0x16
- #define ICM20948_I2C_SLV4_DI_ADDR 0x17
- #define ICM20948_REG_BANK_SEL_BANK0_VALUE 0x00
- #define ICM20948_REG_BANK_SEL_BANK1_VALUE 0x10
- #define ICM20948_REG_BANK_SEL_BANK2_VALUE 0x20
- #define ICM20948_REG_BANK_SEL_BANK3_VALUE 0x30
- #define ICM20948_BANK0_WHO_AM_I_VALUE 0xEA
- #define AK09916_CNTL2_ADDR 0x31
- #define ICM20948_INT_ENABLE_ADDR 0x10
- #define ICM20948_INT_ENABLE_1_ADDR 0x11
- #define ICM20948_INT_ENABLE_2_ADDR 0x12
- #define ICM20948_INT_ENABLE_3_ADDR 0x13

### 7.10.1 Macro Definition Documentation

#### 7.10.1.1 AK09916_CNTL2_ADDR

```
#define AK09916_CNTL2_ADDR 0x31
```

#### 7.10.1.2 ICM20948_ACCEL_CONFIG_1_ADDR

```
#define ICM20948_ACCEL_CONFIG_1_ADDR 0x14
```

#### 7.10.1.3 ICM20948_ACCEL_CONFIG_1_BANK

```
#define ICM20948_ACCEL_CONFIG_1_BANK 2
```

#### 7.10.1.4 ICM20948_ACCEL_INTEL_CTRL_ADDR

```
#define ICM20948_ACCEL_INTEL_CTRL_ADDR 0x12
```

#### 7.10.1.5 ICM20948_ACCEL_INTEL_CTRL_BANK

```
#define ICM20948_ACCEL_INTEL_CTRL_BANK 2
```

#### 7.10.1.6 ICM20948_ACCEL_OUT_BANK

```
#define ICM20948_ACCEL_OUT_BANK 0
```

#### 7.10.1.7 ICM20948_ACCEL_SMPLRT_DIV_1_ADDR

```
#define ICM20948_ACCEL_SMPLRT_DIV_1_ADDR 0x10
```

#### 7.10.1.8 ICM20948_ACCEL_SMPLRT_DIV_1_BANK

```
#define ICM20948_ACCEL_SMPLRT_DIV_1_BANK 2
```

#### 7.10.1.9 ICM20948_ACCEL_SMPLRT_DIV_2_ADDR

```
#define ICM20948_ACCEL_SMPLRT_DIV_2_ADDR 0x11
```

#### 7.10.1.10 ICM20948_ACCEL_SMPLRT_DIV_2_BANK

```
#define ICM20948_ACCEL_SMPLRT_DIV_2_BANK 2
```

#### 7.10.1.11 ICM20948_ACCEL_WOM_THR_ADDR

```
#define ICM20948_ACCEL_WOM_THR_ADDR 0x13
```

#### 7.10.1.12 ICM20948_ACCEL_WOM_THR_BANK

```
#define ICM20948_ACCEL_WOM_THR_BANK 2
```

#### 7.10.1.13 ICM20948_ACCEL_XOUT_H_ADDR

```
#define ICM20948_ACCEL_XOUT_H_ADDR 0x2D
```

#### 7.10.1.14 ICM20948_ACCEL_XOUT_L_ADDR

```
#define ICM20948_ACCEL_XOUT_L_ADDR 0x2E
```

#### 7.10.1.15 ICM20948_ACCEL_YOUT_H_ADDR

```
#define ICM20948_ACCEL_YOUT_H_ADDR 0x2F
```

#### 7.10.1.16 ICM20948_ACCEL_YOUT_L_ADDR

```
#define ICM20948_ACCEL_YOUT_L_ADDR 0x30
```

#### 7.10.1.17 ICM20948_ACCEL_ZOUT_H_ADDR

```
#define ICM20948_ACCEL_ZOUT_H_ADDR 0x31
```

#### 7.10.1.18 ICM20948_ACCEL_ZOUT_L_ADDR

```
#define ICM20948_ACCEL_ZOUT_L_ADDR 0x32
```

#### 7.10.1.19 ICM20948_BANK0_WHO_AM_I_VALUE

```
#define ICM20948_BANK0_WHO_AM_I_VALUE 0xEA
```

#### 7.10.1.20 ICM20948_EXT_SLV_SENS_DATA_00_ADDR

```
#define ICM20948_EXT_SLV_SENS_DATA_00_ADDR 0x3B
```

#### 7.10.1.21 ICM20948_EXT_SLV_SENS_DATA_00_BANK

```
#define ICM20948_EXT_SLV_SENS_DATA_00_BANK 0
```

#### 7.10.1.22 ICM20948_GYRO_CONFIG_1_ADDR

```
#define ICM20948_GYRO_CONFIG_1_ADDR 0x01
```

#### 7.10.1.23 ICM20948_GYRO_CONFIG_1_BANK

```
#define ICM20948_GYRO_CONFIG_1_BANK 2
```

#### 7.10.1.24 ICM20948_GYRO_OUT_BANK

```
#define ICM20948_GYRO_OUT_BANK 0
```

#### 7.10.1.25 ICM20948_GYRO_SMPLRT_DIV_ADDR

```
#define ICM20948_GYRO_SMPLRT_DIV_ADDR 0x00
```

#### 7.10.1.26 ICM20948_GYRO_SMPLRT_DIV_BANK

```
#define ICM20948_GYRO_SMPLRT_DIV_BANK 2
```

#### 7.10.1.27 ICM20948_GYRO_XOUT_H_ADDR

```
#define ICM20948_GYRO_XOUT_H_ADDR 0x33
```

#### 7.10.1.28 ICM20948_GYRO_XOUT_L_ADDR

```
#define ICM20948_GYRO_XOUT_L_ADDR 0x34
```

#### 7.10.1.29 ICM20948_GYRO_YOUT_H_ADDR

```
#define ICM20948_GYRO_YOUT_H_ADDR 0x35
```

#### 7.10.1.30 ICM20948_GYRO_YOUT_L_ADDR

```
#define ICM20948_GYRO_YOUT_L_ADDR 0x36
```

#### 7.10.1.31 ICM20948_GYRO_ZOUT_H_ADDR

```
#define ICM20948_GYRO_ZOUT_H_ADDR 0x37
```

#### 7.10.1.32 ICM20948_GYRO_ZOUT_L_ADDR

```
#define ICM20948_GYRO_ZOUT_L_ADDR 0x38
```

#### 7.10.1.33 ICM20948_I2C_ADDR

```
#define ICM20948_I2C_ADDR 0x69
```

#### 7.10.1.34 ICM20948_I2C_MST_CTRL_ADDR

```
#define ICM20948_I2C_MST_CTRL_ADDR 0x01
```

#### 7.10.1.35 ICM20948_I2C_MST_CTRL_BANK

```
#define ICM20948_I2C_MST_CTRL_BANK 3
```

#### 7.10.1.36 ICM20948_I2C_MST_STATUS_ADDR

```
#define ICM20948_I2C_MST_STATUS_ADDR 0x17
```

#### 7.10.1.37 ICM20948_I2C_MST_STATUS_BANK

```
#define ICM20948_I2C_MST_STATUS_BANK 0
```

#### 7.10.1.38 ICM20948_I2C_SLV0_ADDR_ADDR

```
#define ICM20948_I2C_SLV0_ADDR_ADDR 0x03
```

#### 7.10.1.39 ICM20948_I2C_SLV0_ADDR_BANK

```
#define ICM20948_I2C_SLV0_ADDR_BANK 3
```

#### 7.10.1.40 ICM20948_I2C_SLV0_CTRL_ADDR

```
#define ICM20948_I2C_SLV0_CTRL_ADDR 0x05
```

#### 7.10.1.41 ICM20948_I2C_SLV0_CTRL_BANK

```
#define ICM20948_I2C_SLV0_CTRL_BANK 3
```

#### 7.10.1.42 ICM20948_I2C_SLV0_REG_ADDR

```
#define ICM20948_I2C_SLV0_REG_ADDR 0x04
```

#### 7.10.1.43 ICM20948_I2C_SLV0_REG_BANK

```
#define ICM20948_I2C_SLV0_REG_BANK 3
```

#### 7.10.1.44 ICM20948_I2C_SLV4_ADDR_ADDR

```
#define ICM20948_I2C_SLV4_ADDR_ADDR 0x13
```

#### 7.10.1.45 ICM20948_I2C_SLV4_ADDR_BANK

```
#define ICM20948_I2C_SLV4_ADDR_BANK 3
```

#### 7.10.1.46 ICM20948_I2C_SLV4_CTRL_ADDR

```
#define ICM20948_I2C_SLV4_CTRL_ADDR 0x15
```

#### 7.10.1.47 ICM20948_I2C_SLV4_CTRL_BANK

```
#define ICM20948_I2C_SLV4_CTRL_BANK 3
```

#### 7.10.1.48 ICM20948_I2C_SLV4_DI_ADDR

```
#define ICM20948_I2C_SLV4_DI_ADDR 0x17
```

#### 7.10.1.49 ICM20948_I2C_SLV4_DI_BANK

```
#define ICM20948_I2C_SLV4_DI_BANK 3
```

#### 7.10.1.50 ICM20948_I2C_SLV4_DO_ADDR

```
#define ICM20948_I2C_SLV4_DO_ADDR 0x16
```

#### 7.10.1.51 ICM20948_I2C_SLV4_DO_BANK

```
#define ICM20948_I2C_SLV4_DO_BANK 3
```

#### 7.10.1.52 ICM20948_I2C_SLV4_REG_ADDR

```
#define ICM20948_I2C_SLV4_REG_ADDR 0x14
```

#### 7.10.1.53 ICM20948_I2C_SLV4_REG_BANK

```
#define ICM20948_I2C_SLV4_REG_BANK 3
```

#### 7.10.1.54 ICM20948_INT_ENABLE_1_ADDR [1/2]

```
#define ICM20948_INT_ENABLE_1_ADDR 0x11
```

#### 7.10.1.55 ICM20948_INT_ENABLE_1_ADDR [2/2]

```
#define ICM20948_INT_ENABLE_1_ADDR 0x11
```

#### 7.10.1.56 ICM20948_INT_ENABLE_1_BANK

```
#define ICM20948_INT_ENABLE_1_BANK 0
```

#### 7.10.1.57 ICM20948_INT_ENABLE_2_ADDR

```
#define ICM20948_INT_ENABLE_2_ADDR 0x12
```

#### 7.10.1.58 ICM20948_INT_ENABLE_2_BANK

```
#define ICM20948_INT_ENABLE_2_BANK 0
```

#### 7.10.1.59 ICM20948_INT_ENABLE_3_ADDR

```
#define ICM20948_INT_ENABLE_3_ADDR 0x13
```

#### 7.10.1.60 ICM20948_INT_ENABLE_3_BANK

```
#define ICM20948_INT_ENABLE_3_BANK 0
```

#### 7.10.1.61 ICM20948_INT_ENABLE_ADDR [1/2]

```
#define ICM20948_INT_ENABLE_ADDR 0x10
```

#### 7.10.1.62 ICM20948_INT_ENABLE_ADDR [2/2]

```
#define ICM20948_INT_ENABLE_ADDR 0x10
```

#### 7.10.1.63 ICM20948_INT_ENABLE_BANK

```
#define ICM20948_INT_ENABLE_BANK 0
```

#### 7.10.1.64 ICM20948_INT_PIN_CFG_ADDR

```
#define ICM20948_INT_PIN_CFG_ADDR 0x0F
```

#### 7.10.1.65 ICM20948_INT_PIN_CFG_BANK

```
#define ICM20948_INT_PIN_CFG_BANK 0
```

#### 7.10.1.66 ICM20948_INT_STATUS_1_ADDR

```
#define ICM20948_INT_STATUS_1_ADDR 0x1A
```

#### 7.10.1.67 ICM20948_INT_STATUS_1_BANK

```
#define ICM20948_INT_STATUS_1_BANK 0
```

#### 7.10.1.68 ICM20948_INT_STATUS_ADDR

```
#define ICM20948_INT_STATUS_ADDR 0x19
```

#### 7.10.1.69 ICM20948_INT_STATUS_BANK

```
#define ICM20948_INT_STATUS_BANK 0
```

#### 7.10.1.70 ICM20948_LP_CONFIG_ADDR

```
#define ICM20948_LP_CONFIG_ADDR 0x05
```

#### 7.10.1.71 ICM20948_LP_CONFIG_BANK

```
#define ICM20948_LP_CONFIG_BANK 0
```

#### 7.10.1.72 ICM20948_MAGN_I2C_ADDR

```
#define ICM20948_MAGN_I2C_ADDR 0x0C
```

#### 7.10.1.73 ICM20948_PWR_MGMT_1_ADDR

```
#define ICM20948_PWR_MGMT_1_ADDR 0x06
```

#### 7.10.1.74 ICM20948_PWR_MGMT_1_BANK

```
#define ICM20948_PWR_MGMT_1_BANK 0
```

#### 7.10.1.75 ICM20948_PWR_MGMT_2_ADDR

```
#define ICM20948_PWR_MGMT_2_ADDR 0x07
```

#### 7.10.1.76 ICM20948_PWR_MGMT_2_BANK

```
#define ICM20948_PWR_MGMT_2_BANK 0
```

#### 7.10.1.77 ICM20948_REG_BANK_SEL_ADDR

```
#define ICM20948_REG_BANK_SEL_ADDR 0x7F
```

#### 7.10.1.78 ICM20948_REG_BANK_SEL_BANK0_VALUE

```
#define ICM20948_REG_BANK_SEL_BANK0_VALUE 0x00
```

#### 7.10.1.79 ICM20948_REG_BANK_SEL_BANK1_VALUE

```
#define ICM20948_REG_BANK_SEL_BANK1_VALUE 0x10
```

#### 7.10.1.80 ICM20948_REG_BANK_SEL_BANK2_VALUE

```
#define ICM20948_REG_BANK_SEL_BANK2_VALUE 0x20
```

#### 7.10.1.81 ICM20948_REG_BANK_SEL_BANK3_VALUE

```
#define ICM20948_REG_BANK_SEL_BANK3_VALUE 0x30
```

#### 7.10.1.82 ICM20948_TEMP_OUT_BANK

```
#define ICM20948_TEMP_OUT_BANK 0
```

#### 7.10.1.83 ICM20948_TEMP_OUT_H_ADDR

```
#define ICM20948_TEMP_OUT_H_ADDR 0x39
```

#### 7.10.1.84 ICM20948_TEMP_OUT_L_ADDR

```
#define ICM20948_TEMP_OUT_L_ADDR 0x3A
```

#### 7.10.1.85 ICM20948_USER_CTRL_ADDR

```
#define ICM20948_USER_CTRL_ADDR 0x03
```

#### 7.10.1.86 ICM20948_USER_CTRL_BANK

```
#define ICM20948_USER_CTRL_BANK 0
```

#### 7.10.1.87 ICM20948_WHO_AM_I_ADDR

```
#define ICM20948_WHO_AM_I_ADDR 0x00
```

#### 7.10.1.88 ICM20948_WHO_AM_I_BANK

```
#define ICM20948_WHO_AM_I_BANK 0
```

## 7.11 icm20948_defs.hpp

Go to the documentation of this file.

```
00001 #ifndef ICM20948_REGS_HPP
00002 #define ICM20948_REGS_HPP
00003
00004 // Generic defines
00005 #define ICM20948_I2C_ADDR       0x69
00006 #define ICM20948_MAGN_I2C_ADDR  0x0C
00007
00008 // Banks
00009 // Bank 0
00010 #define ICM20948_WHO_AM_I_BANK              0
00011
00012 #define ICM20948_USER_CTRL_BANK             0
00013
00014 #define ICM20948_LP_CONFIG_BANK             0
00015
00016 #define ICM20948_PWR_MGMT_1_BANK            0
00017 #define ICM20948_PWR_MGMT_2_BANK            0
00018
00019 #define ICM20948_INT_PIN_CFG_BANK           0
00020
00021 #define ICM20948_I2C_MST_STATUS_BANK        0
00022
00023 #define ICM20948_ACCEL_OUT_BANK             0
00024 #define ICM20948_GYRO_OUT_BANK              0
00025 #define ICM20948_TEMP_OUT_BANK              0
00026 #define ICM20948_EXT_SLV_SENS_DATA_00_BANK  0
00027 #define ICM20948_INT_ENABLE_BANK            0
00028 #define ICM20948_INT_ENABLE_1_BANK          0
00029 #define ICM20948_INT_ENABLE_2_BANK          0
00030 #define ICM20948_INT_ENABLE_3_BANK          0
00031
00032 #define ICM20948_INT_STATUS_BANK            0
00033 #define ICM20948_INT_STATUS_1_BANK          0
00034
00035
00036 // Bank 2
00037 #define ICM20948_GYRO_SMPLRT_DIV_BANK       2
```

```
00038 #define ICM20948_GYRO_CONFIG_1_BANK         2
00039 #define ICM20948_ACCEL_SMPLRT_DIV_1_BANK    2
00040 #define ICM20948_ACCEL_SMPLRT_DIV_2_BANK    2
00041 #define ICM20948_ACCEL_CONFIG_1_BANK        2
00042 #define ICM20948_ACCEL_INTEL_CTRL_BANK      2
00043 #define ICM20948_ACCEL_WOM_THR_BANK         2
00044
00045 // Bank 3
00046 #define ICM20948_I2C_MST_CTRL_BANK          3
00047 #define ICM20948_I2C_SLV0_ADDR_BANK         3
00048 #define ICM20948_I2C_SLV0_REG_BANK          3
00049 #define ICM20948_I2C_SLV0_CTRL_BANK         3
00050 #define ICM20948_I2C_SLV4_ADDR_BANK         3
00051 #define ICM20948_I2C_SLV4_REG_BANK          3
00052 #define ICM20948_I2C_SLV4_CTRL_BANK         3
00053 #define ICM20948_I2C_SLV4_DO_BANK           3
00054 #define ICM20948_I2C_SLV4_DI_BANK           3
00055
00056
00057
00058 // Addresses
00059 #define ICM20948_REG_BANK_SEL_ADDR          0x7F
00060
00061 // Addresses: Bank 0
00062 #define ICM20948_WHO_AM_I_ADDR              0x00
00063
00064 #define ICM20948_USER_CTRL_ADDR             0x03
00065
00066 #define ICM20948_LP_CONFIG_ADDR             0x05
00067
00068 #define ICM20948_PWR_MGMT_1_ADDR            0x06
00069 #define ICM20948_PWR_MGMT_2_ADDR            0x07
00070
00071 #define ICM20948_INT_PIN_CFG_ADDR           0x0F
00072
00073 #define ICM20948_I2C_MST_STATUS_ADDR        0x17
00074
00075 #define ICM20948_ACCEL_XOUT_H_ADDR          0x2D
00076 #define ICM20948_ACCEL_XOUT_L_ADDR          0x2E
00077 #define ICM20948_ACCEL_YOUT_H_ADDR          0x2F
00078 #define ICM20948_ACCEL_YOUT_L_ADDR          0x30
00079 #define ICM20948_ACCEL_ZOUT_H_ADDR          0x31
00080 #define ICM20948_ACCEL_ZOUT_L_ADDR          0x32
00081
00082 #define ICM20948_GYRO_XOUT_H_ADDR           0x33
00083 #define ICM20948_GYRO_XOUT_L_ADDR           0x34
00084 #define ICM20948_GYRO_YOUT_H_ADDR           0x35
00085 #define ICM20948_GYRO_YOUT_L_ADDR           0x36
00086 #define ICM20948_GYRO_ZOUT_H_ADDR           0x37
00087 #define ICM20948_GYRO_ZOUT_L_ADDR           0x38
00088
00089 #define ICM20948_TEMP_OUT_H_ADDR            0x39
00090 #define ICM20948_TEMP_OUT_L_ADDR            0x3A
00091
00092 #define ICM20948_EXT_SLV_SENS_DATA_00_ADDR  0x3B
00093
00094 #define ICM20948_INT_ENABLE_ADDR            0x10
00095 #define ICM20948_INT_ENABLE_1_ADDR          0x11
00096 #define ICM20948_INT_STATUS_ADDR            0x19
00097 #define ICM20948_INT_STATUS_1_ADDR          0x1A
00098
00099 // Addresses: Bank 2
00100 #define ICM20948_GYRO_SMPLRT_DIV_ADDR       0x00
00101 #define ICM20948_GYRO_CONFIG_1_ADDR         0x01
00102 #define ICM20948_ACCEL_SMPLRT_DIV_1_ADDR    0x10
00103 #define ICM20948_ACCEL_SMPLRT_DIV_2_ADDR    0x11
00104 #define ICM20948_ACCEL_CONFIG_1_ADDR        0x14
00105
00106 #define ICM20948_ACCEL_INTEL_CTRL_ADDR      0x12
00107 #define ICM20948_ACCEL_WOM_THR_ADDR         0x13
00108
00109 // Addresses: Bank 3
00110 #define ICM20948_I2C_MST_CTRL_ADDR          0x01
00111 #define ICM20948_I2C_SLV0_ADDR_ADDR         0x03
00112 #define ICM20948_I2C_SLV0_REG_ADDR          0x04
00113 #define ICM20948_I2C_SLV0_CTRL_ADDR         0x05
00114 #define ICM20948_I2C_SLV4_ADDR_ADDR         0x13
00115 #define ICM20948_I2C_SLV4_REG_ADDR          0x14
00116 #define ICM20948_I2C_SLV4_CTRL_ADDR         0x15
00117 #define ICM20948_I2C_SLV4_DO_ADDR           0x16
00118 #define ICM20948_I2C_SLV4_DI_ADDR           0x17
00119
00120
00121
00122 // Values
00123 #define ICM20948_REG_BANK_SEL_BANK0_VALUE 0x00
00124 #define ICM20948_REG_BANK_SEL_BANK1_VALUE 0x10
```

```
00125 #define ICM20948_REG_BANK_SEL_BANK2_VALUE 0x20
00126 #define ICM20948_REG_BANK_SEL_BANK3_VALUE 0x30
00127
00128 // Values: Bank 0
00129 #define ICM20948_BANK0_WHO_AM_I_VALUE     0xEA
00130
00131
00132
00133 // Magnetometer (AK09916)
00134 #define AK09916_CNTL2_ADDR  0x31
00135
00136 // Interrupt registers
00137 #define ICM20948_INT_ENABLE_ADDR 0x10
00138 #define ICM20948_INT_ENABLE_1_ADDR 0x11
00139 #define ICM20948_INT_ENABLE_2_ADDR 0x12
00140 #define ICM20948_INT_ENABLE_3_ADDR 0x13
00141
00142 #endif
```

## 7.12 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

## 7.13 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
```

## 7.14 src/libs/I2C/include/icm20948_utils.hpp File Reference

```
#include <cstdint>
#include "yaml-cpp/yaml.h"
```

Include dependency graph for icm20948_utils.hpp: This graph shows which files directly or indirectly include this file:

### Classes

- struct icm20948::accel_settings

  *Configuration settings for an accelerometer sensor.*
- struct icm20948::gyro_settings

  *Configuration settings for a gyroscope sensor.*
- struct icm20948::magn_settings

*Configuration settings for a magnetometer sensor.*

- struct icm20948::settings

  *Aggregated configuration settings for sensor modules.*

### Namespaces

- namespace icm20948

### Typedefs

- typedef struct icm20948::accel_settings icm20948::accel_settings
- typedef struct icm20948::gyro_settings icm20948::gyro_settings
- typedef struct icm20948::magn_settings icm20948::magn_settings
- typedef struct icm20948::settings icm20948::settings

### Enumerations

- enum icm20948::accel_scale { icm20948::ACCEL_2G = 0 , icm20948::ACCEL_4G , icm20948::ACCEL_8G , icm20948::ACCEL_16G }

  *Represents the accelerometer full-scale range settings.*
- enum icm20948::accel_dlpf_config { icm20948::ACCEL_DLPF_246HZ = 0 , icm20948::ACCEL_DLPF_246HZ_2 , icm20948::ACCEL_DLPF_111_4HZ , icm20948::ACCEL_DLPF_50_4HZ , icm20948::ACCEL_DLPF_23_9HZ , icm20948::ACCEL_DLPF_11_5HZ , icm20948::ACCEL_DLPF_5_7HZ , icm20948::ACCEL_DLPF_473HZ }

  *Represents the accelerometer Digital Low Pass Filter (DLPF) configuration settings.*
- enum icm20948::gyro_scale { icm20948::GYRO_250DPS = 0 , icm20948::GYRO_500DPS , icm20948::GYRO_1000DPS , icm20948::GYRO_2000DPS }

  *Represents the gyroscope full-scale range settings.*
- enum icm20948::gyro_dlpf_config { icm20948::GYRO_DLPF_196_6HZ = 0 , icm20948::GYRO_DLPF_151_8HZ , icm20948::GYRO_DLPF_119_5HZ , icm20948::GYRO_DLPF_51_2HZ , icm20948::GYRO_DLPF_23_9HZ , icm20948::GYRO_DLPF_11_6HZ , icm20948::GYRO_DLPF_5_7HZ , icm20948::GYRO_DLPF_361_4HZ }

  *Represents the gyroscope Digital Low Pass Filter (DLPF) configuration settings.*
- enum icm20948::magn_mode { icm20948::MAGN_SHUTDOWN = 0 , icm20948::MAGN_SINGLE = 1 , icm20948::MAGN_10HZ = 2 , icm20948::MAGN_20HZ = 4 , icm20948::MAGN_50HZ = 6 , icm20948::MAGN_100HZ = 8 , icm20948::MAGN_SELF_TEST = 16 }

  *Represents the magnetometer operating modes.*

### Functions

- float icm20948::accel_scale_factor (accel_scale scale)

  *Calculates the scale factor for accelerometer measurements.*
- std::string icm20948::accel_scale_to_str (accel_scale scale)

  *Converts an accelerometer scale value to its corresponding string representation.*
- std::string icm20948::accel_dlpf_config_to_str (accel_dlpf_config config)

  *Converts an accelerometer Digital Low-Pass Filter (DLPF) configuration value to its corresponding string representation.*
- float icm20948::gyro_scale_factor (gyro_scale scale)

*Calculates the scale factor for gyroscope measurements.*

- std::string icm20948::gyro_scale_to_str (gyro_scale scale)

  *Converts a gyroscope scale value to its corresponding string representation.*

- std::string icm20948::gyro_dlpf_config_to_str (gyro_dlpf_config config)

  *Converts a gyroscope Digital Low-Pass Filter (DLPF) configuration value to its corresponding string representation.*

- std::string icm20948::magn_mode_to_str (magn_mode mode)

  *Converts a magnetometer operation mode value to its corresponding string representation.*

## 7.15 icm20948_utils.hpp

Go to the documentation of this file.

```
00001 #ifndef ICM2094_UTILS_HPP
00002 #define ICM2094_UTILS_HPP
00003
00004 #include <cstdint>
00005 #include "yaml-cpp/yaml.h"
00006
00007 namespace icm20948
00008 {
00009     /*** ACCELEROMETER SETTINGS TYPEDEFS ***/
00024     typedef enum {ACCEL_2G = 0,
00025                   ACCEL_4G,
00026                   ACCEL_8G,
00027                   ACCEL_16G
00028     } accel_scale;
00029
00049     typedef enum {ACCEL_DLPF_246HZ = 0,
00050                   ACCEL_DLPF_246HZ_2,
00051                   ACCEL_DLPF_111_4HZ,
00052                   ACCEL_DLPF_50_4HZ,
00053                   ACCEL_DLPF_23_9HZ,
00054                   ACCEL_DLPF_11_5HZ,
00055                   ACCEL_DLPF_5_7HZ,
00056                   ACCEL_DLPF_473HZ
00057     } accel_dlpf_config;
00058
00092     typedef struct accel_settings
00093     {
00094         // Range [0, 4095]
00095         uint16_t sample_rate_div;
00096
00097         // Full scale of measurements +/-
00098         accel_scale scale;
00099
00100         // Digital Low-Pass Filter enable
00101         bool dlpf_enable;
00102
00103         // Digital Low-Pass Filter settings
00104         accel_dlpf_config dlpf_config;
00105
00106
00107         accel_settings(uint16_t sample_rate_div      = 0,
00108                        accel_scale scale             = ACCEL_2G,
00109                        bool dlpf_enable              = true,
00110                        accel_dlpf_config dlpf_config = ACCEL_DLPF_246HZ) :
      sample_rate_div(sample_rate_div),
00111                                                                           scale(scale),
00112                                                                           dlpf_enable(dlpf_enable),
00113                                                                           dlpf_config(dlpf_config)
      {};
00114     } accel_settings;
00115
00116
00117
00118     /*** GYROSCOPE SETTINGS TYPEDEFS ***/
00133     typedef enum {GYRO_250DPS = 0,
00134                   GYRO_500DPS,
00135                   GYRO_1000DPS,
00136                   GYRO_2000DPS
00137     } gyro_scale;
00138
00158     typedef enum {GYRO_DLPF_196_6HZ = 0,
00159                   GYRO_DLPF_151_8HZ,
00160                   GYRO_DLPF_119_5HZ,
00161                   GYRO_DLPF_51_2HZ,
00162                   GYRO_DLPF_23_9HZ,
```

```
00163                   GYRO_DLPF_11_6HZ,
00164                   GYRO_DLPF_5_7HZ,
00165                   GYRO_DLPF_361_4HZ
00166     } gyro_dlpf_config;
00167
00201     typedef struct gyro_settings
00202     {
00203         // Range [0, 250]
00204         uint8_t sample_rate_div;
00205
00206         // Full scale of measurements +/-
00207         gyro_scale scale;
00208
00209         // Digital Low-Pass Filter enable
00210         bool dlpf_enable;
00211
00212         // Digital Low-Pass Filter settings
00213         gyro_dlpf_config dlpf_config;
00214
00215
00216         gyro_settings(uint8_t sample_rate_div    = 0,
00217                       gyro_scale scale           = GYRO_250DPS,
00218                       bool dlpf_enable           = true,
00219                       gyro_dlpf_config dlpf_config = GYRO_DLPF_196_6HZ) :
      sample_rate_div(sample_rate_div),
00220                                                                          scale(scale),
00221                                                                          dlpf_enable(dlpf_enable),
00222                                                                          dlpf_config(dlpf_config) {};
00223     } gyro_settings;
00224
00225
00226
00227     /*** MAGNETOMETER SETTINGS TYPEDEFS ***/
00245     typedef enum {MAGN_SHUTDOWN = 0,
00246                   MAGN_SINGLE = 1,
00247                   MAGN_10HZ = 2,
00248                   MAGN_20HZ = 4,
00249                   MAGN_50HZ = 6,
00250                   MAGN_100HZ = 8,
00251                   MAGN_SELF_TEST = 16
00252     } magn_mode;
00253
00272     typedef struct magn_settings
00273     {
00274         // Magnetometer operation mode
00275         magn_mode mode;
00276
00277         magn_settings(magn_mode mode = MAGN_100HZ) : mode(mode) {};
00278     } magn_settings;
00279
00280
00281
00282     /*** SENSOR SETTINGS TYPEDEFS ***/
00314     typedef struct settings
00315     {
00316         accel_settings accel;
00317         gyro_settings  gyro;
00318         magn_settings  magn;
00319
00320
00321         settings(accel_settings accel = accel_settings(),
00322                  gyro_settings gyro   = gyro_settings(),
00323                  magn_settings magn   = magn_settings()) : accel(accel),
00324                                                            gyro(gyro),
00325                                                            magn(magn) {};
00326
00342         settings(YAML::Node config_file_node);
00343     } settings;
00344
00345
00346
00347
00348     /*** METHODS ***/
00366     float accel_scale_factor(accel_scale scale);
00367
00382     std::string accel_scale_to_str(accel_scale scale);
00383
00401     std::string accel_dlpf_config_to_str(accel_dlpf_config config);
00402
00403
00421     float gyro_scale_factor(gyro_scale scale);
00422
00437     std::string gyro_scale_to_str(gyro_scale scale);
00438
00456     std::string gyro_dlpf_config_to_str(gyro_dlpf_config config);
00457
00458
```

```
00474     std::string magn_mode_to_str(magn_mode mode);
00475 }
00476
00477 #endif
```

## 7.16 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

## 7.17 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
```

## 7.18 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

## 7.19 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
```