# Interactive Audio Visualizer

Generated by Doxygen 1.9.4

# Chapter 1

# Interactive Audio Visualizer

#### 1.0.0.1 Bringing Sound to Life with Interactive Real-Time Audio-Visual Experience

### 1.0.1 Contents



### 1.0.2 Description

This project develops an innovative interactive tool designed to offer a real-time audiovisual experience for users. The envisioned objective of IAV is to provide individuals with limited mobility or minimal musical background an accessible way to engage with positive musical stimuli, using simple actions like hand movements, to inspire and motivate them to participate in creative expression.

### 1.0.3 Documentation

You can access the full documentation for the project in PDF format by clicking here.

If you'd like to generate the documentation in different formats using **Doxygen**, run the following command in your terminal:

```
cd interactive-audio-visualizer
doxygen files/documentation/Doxyfile
```

You may then find the generated documentation in the `files/documentation/docs` folder.

A UML class diagram that illustrates the structure of all the classes in the project is available for reference. You can view it by clicking the following link: `UML Class Diagram Image` or check it online using the `draw.io online viewer`.

### 1.0.4 Build

Before building, the following dependencies must be installed on your system:

- `jack audio`
- `opencv`
- `fftw3`
- `samplerate`
- `asound2`
- `libx11`
- `xrandr`
- `qt6`
- `sqlite3`
- `v4l`

To compile the software, follow these steps:

1. Navigate to the project directory:
   ```
   cd interactive-audio-visualizer
   ```
2. Create and enter a build directory:
   ```
   mkdir build && cd build
   ```
3. Generate the build files:
   ```
   cmake ..
   ```
   4.Compile the project:
   ```
   cmake --build .
   ```

**Additional Configuration Options** You can customize the build process with the following CMake options:

- To build the tests:
  ```
  cmake -DBUILD_TESTS=ON ..
  ```
- To enable static analysis using cppcheck and clang-tidy:
  ```
  cmake -DSTATIC_ANALYSIS=ON ..
  ```

### 1.0.5 Docker

To build the Docker image, execute the following command in the project directory:

1. Build the Docker image:

```
cd interactive-audio-visualizer
docker build -t iav:latest .
```

1. Run the Docker container:

```
xhost +local:docker
docker run --rm --privileged --security-opt seccomp=unconfined -e DISPLAY=$DISPLAY -v
      /tmp/.X11-unix:/tmp/.X11-unix --device /dev/snd -v /dev:/dev -v
      $(pwd)/data:/home/iav/interactive-audio-visualizer/data iav:latest
xhost -local:docker
```

### 1.0.6 Usage

#### 1.0.6.1 Prerequisites

Before running the Interactive Audio Visualizer application, ensure that the following hardware components are connected to your PC:

- a webcam
- an audio output device (e.g., speakers or headphones)
- a display screen (obviously!)

The application will automatically detect available hardware before opening the settings menu. The application will automatically detect available hardware components before launching the settings menu, upon running the compiled executable.

#### 1.0.6.2 Launching the Application

To launch the Interactive Audio Visualizer, run the compiled executable:
`./interactive-audio-visualizer`

#### 1.0.6.3 Configuring Settings

Upon launch, you'll be presented with the settings interface window which allows you to configure various hardware-related settings:

##### 1.0.6.3.1 Audio Settings

- Configure the audio output device
- Adjust sample rate
- Set buffer size
- Modify bit quantization factor

#### 1.0.6.3.2 Camera Settings

- Select webcam device
- Adjust resolution
- Set frame rate

#### 1.0.6.3.3 Display Settings

- Configure screen resolution
- Set display frame rate

#### 1.0.6.3.4 IAV Settings

- Frequency Range: Adjust the frequency range for audio generation.
- ROI (Region of Interest): Configure the area of interest for the video tracking.
- Trigger Mode (Currently only default option is available): Select the trigger mode for initiating the audio visualization (/ visual audiolization) pipeline.
- Tracking Algorithm: Choose a tracking algorithm.
- Accuracy / Economy Slider (Currently unavailable): Adjust performance settings

### 1.0.6.4 Starting the Experience

Once you have configured the settings, press the **Start button** to begin the interactive audio-visual experience.. Otherwise, pressing $**$X button$**$ in the top-right corner of the interface will close the application.

### 1.0.6.5 Exiting the Application

To close the application, click the $**$"q" button$**$.

### 1.0.6.6 Application Workflow

1. A 5-second countdown timer initiates the experience.
2. The webcam captures a frame to initialize the tracking algorithm.
3. The interactive experience begins:
    - The tracking algorithm continuously updates the audiolizer.
    - The audiolizer streams audio and shares audio data with the visualizer.
    - The visualizer updates in real-time, responding to both visual and audio inputs.
4. After 10 seconds, the cycle restarts from the beginning.
5. If the tracking algorithm fails, the application automatically restarts the cycle.'

### 1.0.7 Version history

2nd Feb 2025

- [x] Refactor the code to improve readability and maintainability.
- [x] Encapsulate jack audio server in a class, instead of using it through a system call.
- [x] Use Qt for the graphical user interface (GUI).
- [x] Use various libraries (libX11, libxrandr) to detect hardware components (webcam, audio output device, etc.).
- [x] Use SQLite for storing and retrieving settings.
- [x] Make config a thread-safe singleton pattern for configuring the application.
- [x] Thread refactoring to improve performance and resource usage.
- [x] Impletent testing
- [x] Github actions for CI/CD
- [x] Static analysis with cppcheck and clang-tidy
- [x] Add Docker support
- [x] Enrich visualization with spectrogram representation
- [x] Enrich visualization with audio volume levels visualization
- [x] fix bugs and optimizations
  - [x] fix camera mirroring
  - [x] upgrade compiling to CMake
  - [x] comply to the rule of 5

8th Feb 2024

- [x] instructions update and improvement in compilation method

20th Nov 2023

- [x] demo app 0.9

### 1.0.8 Future work

In the future, the app could evolve to focus on more specific and impactful goals, such as aiding music therapy, assisting those with limited mobility, or supporting individuals with little musical background.

### 1.0.9 Feedback

If exciting ideas pop up, like ways to help with music therapy, support people with limited mobility, or assist those with little musical background, I encourage you to fork the project and build upon the existing codebase. Additionaly, feel free to share your insights, improvements, or inspiration on `Gitter`, or contact me directly at `melissaspaschalis@gmail.com`. Contributions and feedback are always welcome!

### 1.0.10 Related

If you liked this project, you may also like:

- Eye Harp - Playing music with the eyes

# 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 AudioHardware Namespace Reference

Audio hardware namespace provides functions to interact with audio hardware devices.

### Classes

- struct Info

  *Structure representing audio hardware information.*

### Functions

- const std::vector< unsigned int > supportedRates ({8000, 11025, 16000, 22050, 32000, 44100, 48000, 88200, 96000, 176000, 192000, 352800, 384000})

  *List of supported sample rates.*
- bool get_audio_device_info (int, int, std::pair< unsigned int, unsigned int > &, unsigned int &)

  *Retrieves information about available audio devices related to a specific audio card.*
- void get_audio_hardware_info (std::vector< Info > &)

  *Retrieves information about available audio cards and their supported audio devices.*

### Variables

- const short int MAX_POTENTIAL_AUDIO_DEVICES = 32

  *Custom defined - maximum number of potential audio devices that can be retrieved.*
- constexpr int quantizationRatio { sizeof(float) * CHAR_BIT }

  *Bit size of the floating-point samples in bits.*

### 5.1.1 Detailed Description

Audio hardware namespace provides functions to interact with audio hardware devices.

**Note**

This namespace encapsulates the functionality to retrieve information about available audio devices, sample rates, and number of channels.

### 5.1.2 Function Documentation

#### 5.1.2.1 get_audio_device_info()

```
bool AudioHardware::get_audio_device_info (
            int card,
            int device,
            std::pair< unsigned int, unsigned int > & sample_rate,
            unsigned int & numChannels )
```

Retrieves information about available audio devices related to a specific audio card.

**Parameters**

| | |
|---|---|
| *int* | card - audio card index |
| *int* | device - audio device index |
| *std::pair<unsigned* | int, unsigned int> &sample_rate - sample rate range (min, max) |
| *unsigned* | int &numChannels - number of audio channels |

**Returns**

bool - true if information related to the specific audio card is successfully retrieved

Definition at line 8 of file audiohw.cpp.

```
00009 {
00010     snd_pcm_t *handle;
00011     snd_pcm_hw_params_t *params;
00012     int err;
00013     char name[32];
00014     unsigned int sample_rate_min,
00015                  sample_rate_max;
00016
00017     // Open the PCM device
00018     sprintf(name, "hw:%d,%d", card, device);
00019     err = snd_pcm_open(&handle, name, SND_PCM_STREAM_PLAYBACK, 0);
00020     if (err < 0) {
00021         // Error opening PCM device
00022         return false;
00023     }
00024
00025     // Allocate hardware parameters object
00026     snd_pcm_hw_params_alloca(&params);
00027
00028     // Initialize hwparams with full configuration space
00029     err = snd_pcm_hw_params_any(handle, params);
00030     if (err < 0) {
00031         // Error setting hwparams
00032         snd_pcm_close(handle);
00033         return false;
00034     }
00035
00036     // Get sample rate range
00037     err = snd_pcm_hw_params_get_rate_min(params, &sample_rate_min, nullptr);
00038     if (err < 0) {
00039         // Error getting sample rate min
00040         snd_pcm_close(handle);
00041         return false;
00042     }
00043     err = snd_pcm_hw_params_get_rate_max(params, &sample_rate_max, nullptr);
00044     if (err < 0) {
00045         // Error getting sample rate max
00046         snd_pcm_close(handle);
00047         return false;
00048     }
00049
00050     sample_rate.first = sample_rate_min;
00051     sample_rate.second = sample_rate_max;
00052
00053     // get number of output channels
00054     err = snd_pcm_hw_params_get_channels(params, &numChannels);// channels now holds the number of
      channels (outputs)
00055     if (err < 0 || numChannels ==0 ) {
00056         // Set the desired number of channels (e.g., 2 for stereo)
00057         unsigned int atLeastStereo = 2;
00058         err = snd_pcm_hw_params_set_channels(handle, params, atLeastStereo);
00059         if (err < 0 || numChannels ==0 ) {
00060
00061             unsigned int atLeastMono = 1;
00062             err = snd_pcm_hw_params_set_channels(handle, params, atLeastMono);
00063             if (err < 0 || numChannels ==0) {
```

```
00064                 // Error setting channels
00065                 snd_pcm_close(handle);
00066                 return false;
00067             }
00068             // set numChannels to mono
00069             snd_pcm_hw_params_get_channels(params, &numChannels);
00070         } else {
00071             //set numChannels to stereo
00072             snd_pcm_hw_params_get_channels(params, &numChannels);
00073         }
00074     }
00075
00076     // Close the PCM device
00077     snd_pcm_close(handle);
00078     return true;
00079 }
```

#### 5.1.2.2 get_audio_hardware_info()

```
void AudioHardware::get_audio_hardware_info (
            std::vector< Info > & audio_hw_info )
```

Retrieves information about available audio cards and their supported audio devices.
This function scans the system for connected audio devices and gathers information about each one of them, including its name id, the sample rate range supported and the number of output channels.

**Parameters**

| | |
|---|---|
| *std::vector<Info>* | &audio_hw_info - vector to store audio hardware information |

**Returns**

void

Definition at line 81 of file audiohw.cpp.

```
00081                                                                   {
00082
00083     int card = -1;
00084
00085     // Loop through all available cards
00086     while (true) {
00087
00088         // Find the next card
00089         int err = snd_card_next(&card);
00090         if (err < 0) {
00091             // Error getting next card
00092             break;
00093         }
00094         if (card < 0) {
00095             // No more cards
00096             break;
00097         }
00098
00099         // Open the card control interface
00100         snd_ctl_t *ctl_handle;
00101         char ctl_name[32];
00102         sprintf(ctl_name, "hw:%d", card);
00103         err = snd_ctl_open(&ctl_handle, ctl_name, 0);
00104         if (err < 0) {
00105             // Error opening card
00106             continue;
00107         }
00108
00109         // Get the card info
00110         snd_ctl_card_info_t *info;
00111         snd_ctl_card_info_malloc(&info);
00112         err = snd_ctl_card_info(ctl_handle, info);
00113         if (err < 0) {
00114             // Error getting card info
00115             snd_ctl_close(ctl_handle);
00116             snd_ctl_card_info_free(info);
00117             continue;
00118         }
00119
00120         std::string card_id = snd_ctl_card_info_get_id(info) ;
00121         std::string mixer = snd_ctl_card_info_get_mixername(info);
00122 // card_id = card_id + "("+mixer+")";
00123
```

```
00124         // Check if it is an output device and if it can be opened
00125         snd_pcm_t *handle;
00126         std::string card_name_str ="hw:" + std::to_string(card) + ",0";
00127         const char* card_name = card_name_str.c_str();
00128         if (snd_pcm_open(&handle, card_name, SND_PCM_STREAM_PLAYBACK, 0) >= 0) {
00129             snd_pcm_close(handle);
00130         } else {
00131             continue; // Skip this card if it doesn't support output.
00132         }
00133 
00134         // Free the card info
00135         snd_ctl_card_info_free(info);
00136         snd_ctl_close(ctl_handle);
00137 
00138         // Get PCM device info
00139         int device = 0;
00140         unsigned int numChannels = 0;
00141         std::pair<unsigned int, unsigned int> sample_rate_range;
00142         while(!AudioHardware::get_audio_device_info(card, device,sample_rate_range,numChannels) &&
      device < MAX_POTENTIAL_AUDIO_DEVICES ){
00143             ++device;
00144         }
00145 
00146         Info deviceInfo;
00147         deviceInfo.card_info = std::make_pair(card_id,mixer);
00148         deviceInfo.sample_rate_range = sample_rate_range;
00149         deviceInfo.numberOfChannels = numChannels;
00150         audio_hw_info.push_back(deviceInfo);
00151 
00152     }
00153 }
```

#### 5.1.2.3 supportedRates()

```
const std::vector< unsigned int > AudioHardware::supportedRates (
            {8000, 11025, 16000, 22050, 32000, 44100, 48000, 88200, 96000, 176000, 192000,
352800, 384000}  )
```

List of supported sample rates.

**See also**

for detailed info see : https://en.wikipedia.org/wiki/Sampling_(signal_↩ processing)#Audio_sampling

### 5.1.3 Variable Documentation

#### 5.1.3.1 MAX_POTENTIAL_AUDIO_DEVICES

```
const short int AudioHardware::MAX_POTENTIAL_AUDIO_DEVICES = 32
```

Custom defined - maximum number of potential audio devices that can be retrieved.
Definition at line 16 of file audiohw.h.

#### 5.1.3.2 quantizationRatio

```
constexpr int AudioHardware::quantizationRatio { sizeof(float) * CHAR_BIT }  [constexpr]
```

Bit size of the floating-point samples in bits.
Definition at line 27 of file audiohw.h.

## 5.2 Paths Namespace Reference

Namespace containing the path to settings.db file.

### Variables

- const std::string databasePath {getAbsPath("../data/settings.db")}

### 5.2.1 Detailed Description

Namespace containing the path to settings.db file.

**See also**

getAbsPath() function

### 5.2.2 Variable Documentation

#### 5.2.2.1 databasePath

```
const std::string Paths::databasePath {getAbsPath("../data/settings.db")}
```

Definition at line 25 of file paths.h.

## 5.3 PathsTest Namespace Reference

Namespace containing the path to the test.db file, used for testing purposes.

### Variables

- const std::string databasePath {getAbsPath("../data/test.db")}

### 5.3.1 Detailed Description

Namespace containing the path to the test.db file, used for testing purposes.

**See also**

getAbsPath() function

**Note**

These paths are different from the main paths used in the project. They are used for testing purposes only. The actual paths are defined in paths.h. This separation allows for easier management of paths when running tests.

### 5.3.2 Variable Documentation

#### 5.3.2.1 databasePath

```
const std::string PathsTest::databasePath {getAbsPath("../data/test.db")}
```

Definition at line 36 of file paths.h.

# Chapter 6

# Class Documentation

## 6.1 AudioConfig Struct Reference

Struct to hold audio configuration settings.
`#include <config_types.h>`

### Public Attributes

- std::string audioDevice

  *audioDevice - the name of the audio device.*
- std::atomic< int > sampleRate

  *sampleRate - the sample rate of the audio data.*
- int quantization
- std::atomic< int > bufferSize
- std::atomic< unsigned int > numChannels

### 6.1.1 Detailed Description

Struct to hold audio configuration settings.

**Note**

It uses atomic types for thread-safe access to these values.

This struct is used in the Config class to define audio configuration settings.

**See also**

Config class for managing configuration settings.

audioDevice.- the name of the audio device.

sampleRate - the sample rate of the audio data.

quantization - the quantization of the audio data.

bufferSize - the buffer size for audio data.

numChannels - the number of output audio channels.

Definition at line 18 of file config_types.h.

### 6.1.2 Member Data Documentation

#### 6.1.2.1 audioDevice

`std::string AudioConfig::audioDevice`
audioDevice - the name of the audio device.
Definition at line 20 of file config_types.h.

#### 6.1.2.2 bufferSize

`std::atomic<int> AudioConfig::bufferSize`
Definition at line 26 of file config_types.h.

#### 6.1.2.3 numChannels

`std::atomic<unsigned int> AudioConfig::numChannels`
Definition at line 28 of file config_types.h.

#### 6.1.2.4 quantization

`int AudioConfig::quantization`
Definition at line 24 of file config_types.h.

#### 6.1.2.5 sampleRate

`std::atomic<int> AudioConfig::sampleRate`
sampleRate - the sample rate of the audio data.
Definition at line 22 of file config_types.h.
The documentation for this struct was generated from the following file:

- include/config_types.h

## 6.2 Audiolizer Class Reference

A class responsible for translating tracking signal into audio frequency.
`#include <audiolizer.h>`

### Public Member Functions

- Audiolizer ()
  *Default constructor.*
- bool turn_Image_into_Sound (const bool, const bool, const RegionOfInterest &, Tone &)
  *Member function responsible for obtaining image signal and converting it into sound of a certain frequency.*
- void setAudioUpdater (std::function< void(int, float)>)
  *Method that sets the audio updater function.*

### 6.2.1 Detailed Description

A class responsible for translating tracking signal into audio frequency.

**Note**

This class is responsible for managing the conversion from image to audio.

Definition at line 13 of file audiolizer.h.

## 6.2.2 Constructor & Destructor Documentation

### 6.2.2.1 Audiolizer()

```
Audiolizer::Audiolizer ( )
```

Default constructor.

Definition at line 7 of file audiolizer.cpp.

```
00007                       :cameracfg(Config::getInstance().camconf),iavcfg(Config::getInstance().iavconf){
00008
00009     // @TEMPORARY DISABLED
00010     // init_log_freq_scale(); // currently not used.  Use _int2log_freq (currently not used either) is
      affected by this method..
00011
00012     frequencyRange = iavcfg.maxFrequency - iavcfg.minFrequency;
00013     prev_freq=0;
00014     volume = 0.f;
00015 }
```

## 6.2.3 Member Function Documentation

### 6.2.3.1 setAudioUpdater()

```
void Audiolizer::setAudioUpdater (
            std::function< void(int, float)> func )
```

Method that sets the audio updater function.

**Parameters**

| | |
|---|---|
| *std::function<void(int,float)>* | - the audio updater function that receives as parameters the current frequency and volume. |

**Returns**

void

Definition at line 17 of file audiolizer.cpp.

```
00017                                                                   {
00018     updateAudio = std::move(func);
00019 }
```

### 6.2.3.2 turn_Image_into_Sound()

```
bool Audiolizer::turn_Image_into_Sound (
            const bool tracking_updated,
            const bool pattern_locked,
            const RegionOfInterest & roi,
            Tone & tone )
```

Member function responsible for obtaining image signal and converting it into sound of a certain frequency.

**Parameters**

| | | |
|---|---|---|
| in | *bool* | trackingUpdated - variable that indicates whether there is a new tracking signal. |
| in | *bool* | trackingEnabled - variable that indicates whether the tracking is enabled or not. |
| | *RegionOfInterest&* | roi - variable passed by reference that updates the value of the current tracking signal. |
| in | *Tone&* | - the tone object for storing the current frequency and volume |

**Returns**

bool - returns true if frequency has changed

Definition at line 21 of file audiolizer.cpp.

```
00021
     {
00022
00023 /***
00024 * returns by reference the frequency that will be streamed on the next audio buffer
00025 */
00026
00027     int frequency = tone.frequency.load();
00028     int prevFreq = prev_freq;
00029
00030     if (pattern_locked){
00031         if (tracking_updated)          // if tracking updated --> new x,y --> new freq
00032             translate(roi, frequency);
00033         else{                          // else --> previous frequency
00034             frequency=prev_freq;
00035         }
00036     }else{                            // gradualy fade frequency to zero --> if frequency > 0 , slowly
     decline
00037         if (frequency>1){
00038             gradualy_fade(frequency); // gradualy fade frequency to zero --> if frequency > 0 , slowly
     decline
00039         }else{
00040             frequency=0;
00041             volume = 0.f;
00042         }
00043     }
00044
00045     // update audioStream with the newFrequency
00046     bool frequencyChanged = frequency != prevFreq;
00047     if (frequencyChanged){
00048         updateAudio(frequency , volume);
00049     }
00050     tone.frequency.store(frequency);
00051     tone.volume.store(volume);
00052
00053     return frequencyChanged;
00054
00055 }
```

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

- include/audiolizer.h
- src/audiolizer.cpp

## 6.3 AudioServer Class Reference

The jack-audio server running on the alsa drivers.

`#include <audioserver.h>`

### Public Member Functions

- AudioServer (const char *driverName=supported_driver)

  *Default constructor.*
- ~AudioServer ()

  *Class destructor Destroys the jack audio server and its associated resources.*
- void setup_server ()

  *Setup the jack audio server by changing server parameters and alsa driver parameters.*
- void start_server (std::mutex &, std::condition_variable &, bool &)

  *Starts the jack audio server.*
- void stop_server ()

  *Stops the jack audio server.*
- AudioServer (const AudioServer &)=delete

  *Copy constructor is deleted to prevent accidental use.*
- AudioServer (AudioServer &&)=delete

  *Move constructor is deleted to prevent accidental use.*

- AudioServer & operator= (const AudioServer &)=delete

  *Copy assignment operator is deleted to prevent accidental use.*
- AudioServer & operator= (AudioServer &&)=delete

  *Move assignment operator is deleted to prevent accidental use.*

### 6.3.1 Detailed Description

The jack-audio server running on the alsa drivers.

**Note**

This class encapsulates the functionality for managing the jack audio server and its associated resources It is responsible for setting up the jack audio server, starting it, and stopping it.

The AudioServer class uses the jack library for managing the jack audio server.

**See also**

the jack library documentation : https://jackaudio.org/api/

jack server example : https://github.com/jackaudio/example-clients/blob/master/server_control.c

Definition at line 22 of file audioserver.h.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 AudioServer() [1/3]

```
AudioServer::AudioServer (
            const char * driverName = supported_driver )  [explicit]
```

Default constructor.

Definition at line 10 of file audioserver.cpp.

```
00010                                           :driver_name(driverName),audiocfg
      (Config::getInstance().audconf){
00011     server = jackctl_server_create2(NULL, NULL, NULL);
00012     parameters = jackctl_server_get_parameters(server);
00013     sigmask = jackctl_setup_signals(0);
00014     drivers = jackctl_server_get_drivers_list(server);
00015
00016 }
```

#### 6.3.2.2 ~AudioServer()

```
AudioServer::~AudioServer ( )
```

Class destructor Destroys the jack audio server and its associated resources.

Definition at line 18 of file audioserver.cpp.

```
00018                          {
00019     printf("Stopping server\n");
00020     stop_server();
00021 }
```

#### 6.3.2.3 AudioServer() [2/3]

```
AudioServer::AudioServer (
            const AudioServer &  )  [delete]
```

Copy constructor is deleted to prevent accidental use.

#### 6.3.2.4 AudioServer() [3/3]

```
AudioServer::AudioServer (
            AudioServer &&  )  [delete]
```

Move constructor is deleted to prevent accidental use.

### 6.3.3 Member Function Documentation

#### 6.3.3.1 operator=() [1/2]

```
AudioServer & AudioServer::operator= (
            AudioServer &&  )  [delete]
```

Move assignment operator is deleted to prevent accidental use.

#### 6.3.3.2 operator=() [2/2]

```
AudioServer & AudioServer::operator= (
            const AudioServer &  )  [delete]
```

Copy assignment operator is deleted to prevent accidental use.

#### 6.3.3.3 setup_server()

```
void AudioServer::setup_server ( )
```

Setup the jack audio server by changing server parameters and alsa driver parameters.

**Returns**

void

Definition at line 23 of file audioserver.cpp.

```
00023                                {
00024
00025     change_server_parameters();
00026 #ifdef SERVER_VERBOSE
00027     print_driver_info();
00028 #endif
00029     change_ALSAdriver_parameters();
00030 }
```

#### 6.3.3.4 start_server()

```
void AudioServer::start_server (
            std::mutex & mtx,
            std::condition_variable & cv,
            bool & serverStarted )
```

Starts the jack audio server.

**Parameters**

| | |
|---|---|
| *std::mutex&* | - a mutex object to control synchronization with the client. |
| *std::condition_variable&* | - a condition variable object to signal when the server is ready |

**Returns**

void

**Warning**

The server has to be started before the audio client attempts to connect.

Definition at line 37 of file audioserver.cpp.

```
00037                                                                                {
00038     jackctl_server_open(server, jackctl_server_get_driver());
00039     jackctl_server_start(server);
00040
00041     // Signal that server has been started
00042     {
00043         std::lock_guard<std::mutex> lock(mtx);
00044         serverStarted = true;
00045     }
00046     cv.notify_one();
00047
00048     jackctl_wait_signals(sigmask);
00049 }
```

#### 6.3.3.5 stop_server()

```
void AudioServer::stop_server ( )
```

Stops the jack audio server.

**Returns**

void

Definition at line 31 of file audioserver.cpp.

```
00031                                {
00032     printf("\n\nShutting down server\n\n");
00033     jackctl_server_stop(server);
00034     jackctl_server_close(server);
00035     jackctl_server_destroy(server);
00036 }
```

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

- include/audioserver.h
- src/audioserver.cpp

## 6.4 AudioStream Class Reference

A class representing the audio streaming functionality.

```
#include <audio.h>
```

### Public Member Functions

- AudioStream ()

  *Class contructor. In this program, a setConfig is used for implicit initialization.*
- ~AudioStream ()

  *class destructor disconnects the client from the server*
- void clientConnect (std::mutex &, std::condition_variable &, bool &)

  *Starts a connection to the server and creates the connection graph which connects the inputs with the outputs.*
- void closeStream ()

  *Disconnects the client from the server.*
- int streamBuffer ()

  *Member function for streaming the audio buffer. It is a Callback function called implicitly via the static AudioStream::streamAudio function. Inside the function, the sine wave generator is called to fill the audio buffers with the generated sine waves.*
- void update (int, float)

  *Updates the tone member variable with a new frequency and volume.*
- void setVisualizerUpdater (std::function< void(float)>)

  *Sets the visualizer updater function.*

- AudioStream (const AudioStream &)=delete

  *Copy constructor is deleted to prevent accidental use.*
- AudioStream (AudioStream &&)=delete

  *Move constructor is deleted to prevent accidental use.*
- AudioStream & operator= (const AudioStream &)=delete

  *Copy assignment operator is deleted to prevent accidental use.*
- AudioStream & operator= (AudioStream &&)=delete

  *Move assignment operator is deleted to prevent accidental use.*

### 6.4.1 Detailed Description

A class representing the audio streaming functionality.
Class for routing audio signal. Uses the jack audio API.

**Note**

This class manages the audio streaming process, including connecting to the server, creating the connection graph, and streaming the audio buffer.

**See also**

the jack library documentation : https://jackaudio.org/api/

jack client example : https://github.com/jackaudio/example-clients/blob/master/simple_client.c

Definition at line 22 of file audio.h.

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 AudioStream() [1/3]

```
AudioStream::AudioStream ( )
```

Class contructor. In this program, a setConfig is used for implicit initialization.
Definition at line 18 of file audio.cpp.

```
00018                         :audiocfg (Config::getInstance().audconf){
00019
00020     client_name=clientName;
00021     // nullify all
00022     client = nullptr;
00023     todevice = nullptr;
00024
00025     if (audiocfg.numChannels.load() == 1){
00026         output_ports[1] = nullptr;
00027         outputBuffers[1]=nullptr;
00028         make_sound = &Sine::setMonoSignal;  // Point to setMonoSignal for processing 1 single mono
      buffer
00029     }
00030     else if (audiocfg.numChannels.load() == 2) {
00031         make_sound = &Sine::setStereoSignal;  // Point to setStereoSignal for processing 2 stereo
      buffers
00032     }
00033 }
```

#### 6.4.2.2 ~AudioStream()

```
AudioStream::~AudioStream ( )
```

class destructor disconnects the client from the server
Definition at line 39 of file audio.cpp.

```
00039                           {
00040     closeStream();
00041     std::cout«"Audio stream object destructed"«std::endl;
00042 }
```

#### 6.4.2.3 AudioStream() [2/3]

```
AudioStream::AudioStream (
            const AudioStream &  )  [delete]
```

Copy constructor is deleted to prevent accidental use.

#### 6.4.2.4 AudioStream() [3/3]

```
AudioStream::AudioStream (
            AudioStream &&  )  [delete]
```

Move constructor is deleted to prevent accidental use.

### 6.4.3 Member Function Documentation

#### 6.4.3.1 clientConnect()

```
void AudioStream::clientConnect (
            std::mutex & mtx,
            std::condition_variable & cv,
            bool & serverStarted )
```

Starts a connection to the server and creates the connection graph which connects the inputs with the outputs.

**Parameters**

| | |
|---|---|
| *mutex&* | - mutex for synchronization with the server |
| *condition_variable&* | - condition variable for synchronization with the server |
| *bool&* | - boolean indicating whether the server has started |

**Returns**

void

Definition at line 44 of file audio.cpp.

```
00044                                                                                                                {
00045
00046     std::cout « "Waiting for jack server to start\n";
00047     std::unique_lock<std::mutex> lock(mtx);
00048     cv.wait(lock, [&] { return serverStarted; });
00049
00050     jack_options_t options =
      JackNoStartServer;//(JackSessionID|JackServerName|JackNoStartServer|JackUseExactName|JackNullOption)
00051     jack_status_t status;
00052
00053     /* open a client connection to the JACK server */
00054     client = jack_client_open (client_name, options, &status,nullptr);
00055     if (status & JackNameNotUnique) {    //client name not unique, set a client name;
00056         client_name = jack_get_client_name(client);
00057         std::cerr«"\t»unique name "«client_name«" assigned to the client obj."«std::endl;
00058     }
00059
00060     if (client == NULL) {
00061         std::cerr«"\t»jack_client_open() failed, status = "«status«std::endl;
00062         if (status & JackServerFailed) {
00063             std::cerr«"\t»Unable to connect to JACK server"«std::endl;
00064         }
00065         exit (1);
00066     }
00067     if (status & JackServerStarted) {
00068         std::cout«"\t»JACK server started"«std::endl;
00069     }
00070
00071     //callback
00072     if (jack_set_process_callback (client,streamAudio,this)){ //arg
00073             std::cerr«"\t»Callback operation failed"«std::endl;
00074     }
00075
00076     //prevent failure
```

```
00077     jack_on_shutdown(client,&jack_shutdown,0);
00078
00079     //register physical ports
00080     for (size_t ch=0; ch<audiocfg.numChannels.load();++ch){
00081         std::string portName = (ch%2) ?  ("PortRight"+std::to_string(ch/2)) :
      ("PortLeft"+std::to_string(ch/2));
00082         // std::cout«"portName = "«portName«std::endl;
00083         output_ports[ch]=jack_port_register (client,portName.c_str(),JACK_DEFAULT_AUDIO_TYPE,
      JackPortIsOutput, 0);
00084         // output_port_right=jack_port_register (client,"rightPort",JACK_DEFAULT_AUDIO_TYPE,
      JackPortIsOutput, 0);
00085         if (output_ports[ch] == NULL){
00086             std::cerr«"\t»Unable to register output port for
      {"«jack_port_name(output_ports[ch])«"}"«std::endl;
00087             exit (1);}
00088     }
00089
00090     //activate client
00091     if (jack_activate (client)) {
00092         std::cerr«"\t»cannot activate client {"«client_name«"}"«std::endl;
00093         exit (1);
00094     }
00095
00096     // Getting acces to destination ports
00097     todevice = jack_get_ports (client, NULL, NULL, JackPortIsPhysical|JackPortIsInput);
00098     if (todevice == NULL) {
00099         std::cerr«"\t»no physical playback devices"«std::endl;
00100         exit (1);
00101     }
00102
00103     for (size_t ch=0; ch <audiocfg.numChannels.load();++ch){
00104         if (output_ports[ch]!=NULL){
00105             if (jack_connect (client, jack_port_name(output_ports[ch]), todevice[ch])){//returns full
      name
00106                 std::cerr«"\t»cannot connect left plysical output port {"«todevice[ch]«"} with input
      port {"«jack_port_name(output_ports[ch])«"}"«std::endl;
00107             }
00108         }
00109     }
00110
00111     free (todevice);
00112 }
```

#### 6.4.3.2 closeStream()

```
void AudioStream::closeStream ( )
```

Disconnects the client from the server.

**Returns**

void

Definition at line 115 of file audio.cpp.

```
00115                               {
00116
00117     for (size_t i=0; i<audiocfg.numChannels.load();++i){
00118         if (jack_port_connected(output_ports[i])){
00119             if(jack_port_disconnect(client,output_ports[i])){
00120                 std::cerr«"Couldnt disconnect the "«jack_port_name(output_ports[i])«" output port from
      the main stream"«std::endl;
00121             }
00122         }
00123
00124     }
00125
00126     std::cout«"Closing stream - turning off audio client.."«std::endl;
00127     jack_client_close (client);
00128 }
```

#### 6.4.3.3 operator=() [1/2]

```
AudioStream & AudioStream::operator= (
            AudioStream &&  )  [delete]
```

Move assignment operator is deleted to prevent accidental use.

#### 6.4.3.4 operator=() [2/2]

```
AudioStream & AudioStream::operator= (
            const AudioStream &  )  [delete]
```

Copy assignment operator is deleted to prevent accidental use.

#### 6.4.3.5 setVisualizerUpdater()

```
void AudioStream::setVisualizerUpdater (
            std::function< void(float)> updater )
```

Sets the visualizer updater function.

**Parameters**

| | |
|---|---|
| *std::<void(float)>* | - the visualizer updater function |

**Returns**

void

**Note**

The visualizer updater function takes one by one an audio sample as a float parameter to write on the FIFO-based data structures of Waveform and Spectrogram.

Definition at line 35 of file audio.cpp.

```
00035                                                                    {
00036     sine.setVisualizerUpdater(std::move(updater));
00037 }
```

#### 6.4.3.6 streamBuffer()

```
int AudioStream::streamBuffer ( )
```

Member function for streaming the audio buffer. It is a Callback function called implicitly via the static AudioStream::streamAudio function. Inside the function, the sine wave generator is called to fill the audio buffers with the generated sine waves.

**Returns**

int - success message

Definition at line 130 of file audio.cpp.

```
00130                                {
00131
00132     for (size_t ch = 0 ; ch < audiocfg.numChannels.load(); ++ch){
00133         outputBuffers[ch] = static_cast<float *>(jack_port_get_buffer (output_ports[ch],
      audiocfg.bufferSize.load() ));
00134     }
00135
00136     (sine.*make_sound)(tone,outputBuffers);
00137
00138
00139     return 0;
00140 }
```

#### 6.4.3.7 update()

```
void AudioStream::update (
            int frequency,
            float volume )
```

Updates the tone member variable with a new frequency and volume.

**Parameters**

| | |
|---|---|
| *int* | frequency - the current frequency of the tone |
| *float* | volume - the volume of the tone |

**Returns**

void

Retrieves the minimum and maximum values from the waveform.

**Parameters**

| | | |
|---|---|---|
| out | *frequency* | The tones spectral frequency. |
| out | *volume* | The volume of the tone. |

Definition at line 152 of file audio.cpp.

```
00152                                                {
00153     tone.frequency.store(frequency);
00154     tone.volume.store(volume);
00155 }
```

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

- include/audio.h
- src/audio.cpp

## 6.5 Camera Class Reference

Class representing a camera object.

`#include <camera.h>`

### Public Member Functions

- Camera ()

  *Default constructor.*
- ~Camera ()

  *Destructor for the camera class. It closes the camera and releases any resources.*
- bool capture (cv::Mat &)

  *Method for capturing frames received from the camera.*
- bool frame_elapsed ()

  *Method that make use of std::atomic variable frameToggle to indicate whether a new frame elapsed.*
- Camera (const Camera &)=delete

  *Copy constructor is deleted to prevent accidental use.*
- Camera (Camera &&)=delete

  *Move constructor is deleted to prevent accidental use.*
- Camera & operator= (const Camera &)=delete

  *Copy assignment operator is deleted to prevent accidental use.*
- Camera & operator= (Camera &&)=delete

  *Move assignment operator is deleted to prevent accidental use.*

### 6.5.1 Detailed Description

Class representing a camera object.

**Note**

This class encapsulates the functionality for capturing frames from a camera.

The Camera class is responsible for initializing the camera and capturing frames.

The Camera class uses OpenCV library.

Definition at line 16 of file camera.h.

## 6.5.2 Constructor & Destructor Documentation

### 6.5.2.1 Camera() [1/3]

```
Camera::Camera ( )
```

Default constructor.

Definition at line 6 of file camera.cpp.

```
00006                 :  cameracfg(Config::getInstance().camconf){
00007     frameToggle.store(false);
00008     toggleFrame=false;
00009     cv::Mat frame(cameracfg.camResH.load(),cameracfg.camResW.load(),CV_8UC3);
00010     initialize_camera();
00011 }
```

### 6.5.2.2 ~Camera()

```
Camera::~Camera ( )
```

Destructor for the camera class. It closes the camera and releases any resources.

Definition at line 42 of file camera.cpp.

```
00042                 {
00043     frame.release();
00044     cap.release();
00045     std::cout<<"Camera object destructed"<<std::endl;
00046 }
```

### 6.5.2.3 Camera() [2/3]

```
Camera::Camera (
            const Camera &  )  [delete]
```

Copy constructor is deleted to prevent accidental use.

### 6.5.2.4 Camera() [3/3]

```
Camera::Camera (
            Camera &&  )  [delete]
```

Move constructor is deleted to prevent accidental use.

## 6.5.3 Member Function Documentation

### 6.5.3.1 capture()

```
bool Camera::capture (
            cv::Mat & frame )
```

Method for capturing frames received from the camera.

**Parameters**

| | | |
|---|---|---|
| out | *cv::Mat&* | - the output frame from the camera. |

Definition at line 57 of file camera.cpp.

```
00057                                   {
00058
00059     cap.read(frame);
00060
00061     if(!(frame.empty())){
00062         frameToggle.store(!frameToggle.load());
00063         return true;
00064     }
00065     return false;
00066 }
```

#### 6.5.3.2 frame_elapsed()

```
bool Camera::frame_elapsed ( )
```

Method that make use of std::atomic variable frameToggle to indicate whether a new frame elapsed.

**Returns**

bool - true if a frame has elapsed.

**Note**

This method is never used in this code.

Definition at line 48 of file camera.cpp.

```
00048                                {
00049     atomicChange = frameToggle.load();
00050     if (frameToggle.load()!=toggleFrame){     // process the current input from camera
00051         toggleFrame=atomicChange;
00052         return true;
00053     }else
00054         return false;
00055 }
```

#### 6.5.3.3 operator=() [1/2]

```
Camera & Camera::operator= (
            Camera &&  )  [delete]
```

Move assignment operator is deleted to prevent accidental use.

#### 6.5.3.4 operator=() [2/2]

```
Camera & Camera::operator= (
            const Camera &  )  [delete]
```

Copy assignment operator is deleted to prevent accidental use.
The documentation for this class was generated from the following files:

- include/camera.h
- src/camera.cpp

## 6.6 CameraConfig Struct Reference

Struct to hold camera configuration settings.
`#include <config_types.h>`

### Public Attributes

- std::string device

  *device - the name of the camera device.*
- std::atomic< double > frameRate
- std::atomic< int > camResW
- std::atomic< int > camResH

### 6.6.1 Detailed Description

Struct to hold camera configuration settings.

**Note**

It uses atomic types for thread-safe access to these values.

This struct is used in the Config class to define camera configuration settings.

**See also**

device - the name of the camera device.

frameRate - the frame rate of the camera.

camResW - the width of the camera resolution.

camResH - the height of the camera resolution.

Definition at line 40 of file config_types.h.

### 6.6.2 Member Data Documentation

#### 6.6.2.1 camResH

```
std::atomic<int> CameraConfig::camResH
```

Definition at line 48 of file config_types.h.

#### 6.6.2.2 camResW

```
std::atomic<int> CameraConfig::camResW
```

Definition at line 46 of file config_types.h.

#### 6.6.2.3 device

```
std::string CameraConfig::device
```

device - the name of the camera device.
Definition at line 42 of file config_types.h.

#### 6.6.2.4 frameRate

```
std::atomic<double> CameraConfig::frameRate
```

Definition at line 44 of file config_types.h.
The documentation for this struct was generated from the following file:

- include/config_types.h

## 6.7 CameraInfo Struct Reference

Represents information about a camera.
`#include <camerahw.h>`

### Public Attributes

- std::string devicePath
- std::vector< std::pair< int, int > > resolutions

### 6.7.1 Detailed Description

Represents information about a camera.
This structure contains the device path of the camera and a vector of pairs representing supported resolutions. Each pair contains the width and height of the resolution.

**See also**

getAvailableCameras for more information about retrieving camera information.

Definition at line 15 of file camerahw.h.

### 6.7.2 Member Data Documentation

#### 6.7.2.1 devicePath

```
std::string CameraInfo::devicePath
```

Definition at line 16 of file camerahw.h.

#### 6.7.2.2 resolutions

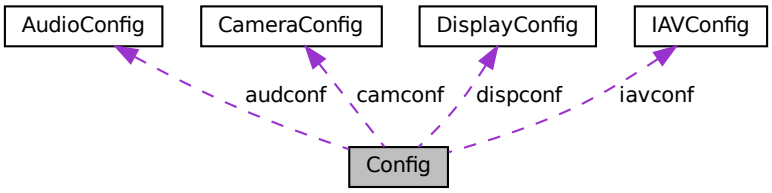
```
std::vector<std::pair<int, int> > CameraInfo::resolutions
```

Definition at line 17 of file camerahw.h.
The documentation for this struct was generated from the following file:

- include/gui/camerahw.h

## 6.8 Config Class Reference

Singleton class to manage configuration settings, providing a unique point of access to the configuration settings.
`#include <config.h>`
Collaboration diagram for Config:




### Public Member Functions

- Config (Config const &)=delete

  *Copy constructor is deleted to fill the requirement of a singleton.*
- void operator= (Config const &)=delete

  *Copy assignment operator is deleted to fill the requirement of a singleton.*
- void display ()

  *Display function. It prints out in the console all the parameter values.*

## Static Public Member Functions

- static Config & getInstance ()

  *Get instance of the Config class. Singleton pattern.*

## Public Attributes

- AudioConfig audconf
- CameraConfig camconf
- DisplayConfig dispconf
- IAVConfig iavconf

### 6.8.1 Detailed Description

Singleton class to manage configuration settings, providing a unique point of access to the configuration settings.

**Note**

The Config class is responsible for reading settings from the settings database and initializing objects of different configuration types.

It uses a map to store settings and provides methods to read and write them.

**See also**

SettingsDB class for managing settings using a SQLite database.

Definition at line 13 of file config.h.

### 6.8.2 Constructor & Destructor Documentation

#### 6.8.2.1 Config()

```
Config::Config (
            Config const &  )  [delete]
```

Copy constructor is deleted to fill the requirement of a singleton.

### 6.8.3 Member Function Documentation

#### 6.8.3.1 display()

```
void Config::display ( )
```

Display function. It prints out in the console all the parameter values.

**Returns**

void

Definition at line 68 of file config.cpp.

```
00068                     {
00069
00070     std::cout«"########## Interactive Audio Visualizer Config ##########\n";
00071     std::cout«"------------------ audio settings ----------------------\n";
00072     std::cout«"audio device             \t:\t"«audconf.audioDevice«std::endl;
00073     std::cout«"sampling rate            \t:\t"«audconf.sampleRate.load()«" samples/sec"«std::endl;
00074     std::cout«"quantization             \t:\t"«audconf.quantization«" bits"«std::endl;
00075     std::cout«"buffer size              \t:\t"«audconf.bufferSize.load()«" samples"«std::endl;
00076     std::cout«"num output channels      \t:\t"«audconf.numChannels.load()«" "«std::endl;
00077     std::cout«"----------------- display settings ---------------------\n";
00078     std::cout«"frames per second        \t:\t"«dispconf.fps.load()«" fps"«std::endl;
00079     std::cout«"display Width            \t:\t"«dispconf.dispResW.load()«" pixels"«std::endl;
00080     std::cout«"display Height           \t:\t"«dispconf.dispResH.load()«" pixels"«std::endl;
```

```
00081     std::cout«"------------------ camera settings ----------------------\n";
00082     std::cout«"camera device            \t:\t"«camconf.device«std::endl;
00083     std::cout«"camera resolution width  \t:\t"«camconf.camResW.load()«" pixels"«std::endl;
00084     std::cout«"camera resolution height \t:\t"«camconf.camResH.load()«" pixels"«std::endl;
00085     std::cout«"camera frame rate        \t:\t"«camconf.frameRate.load()«" fps"«std::endl;
00086     std::cout«"------------------ iav Settings ------------------------\n";
00087     std::cout«"mininum frequency        \t:\t"«iavconf.minFrequency«" Hz"«std::endl;
00088     std::cout«"maxinum frequency        \t:\t"«iavconf.maxFrequency«" Hz"«std::endl;
00089     std::cout«"radius                   \t:\t"«iavconf.roiRadius«" pixels"«std::endl;
00090     std::cout«"trigger method           \t:\t"«iavconf.trigger«std::endl;
00091     std::cout«"tracking algorithm       \t:\t"«iavconf.trackingAlg«std::endl;
00092     // std::cout«"skip frames ratio        \t:\t"«iavconf.skipFramesRatio«std::endl;
00093     // std::cout«"number of skip frames     \t:\t"«iavconf.skipFramesRatio-1«" frames"«std::endl;
00094     std::cout«"############################################################\n\n";
00095
00096 }
```

#### 6.8.3.2 getInstance()

```
static Config & Config::getInstance ( )  [inline], [static]
```

Get instance of the Config class. Singleton pattern.

**Returns**

Config& - a reference to the Config object.

Definition at line 20 of file config.h.

```
00020                                {
00021         static Config config;
00022         return config;
00023     }
```

#### 6.8.3.3 operator=()

```
void Config::operator= (
             Config const &  )  [delete]
```

Copy assignment operator is deleted to fill the requirement of a singleton.

### 6.8.4 Member Data Documentation

#### 6.8.4.1 audconf

```
AudioConfig Config::audconf
```

**See also**

AudioConfig struct for details on audio configuration settings.

Definition at line 43 of file config.h.

#### 6.8.4.2 camconf

```
CameraConfig Config::camconf
```

**See also**

CameraConfig struct for details on camera configuration settings.

Definition at line 48 of file config.h.

#### 6.8.4.3 dispconf

```
DisplayConfig Config::dispconf
```

**See also**

DisplayConfig struct for details on display configuration settings.

Definition at line 53 of file config.h.

#### 6.8.4.4 iavconf

```
IAVConfig Config::iavconf
```

**See also**

IAVConfig struct for details on IAV configuration settings.

Definition at line 58 of file config.h.
The documentation for this class was generated from the following files:

- include/config.h
- src/config.cpp

## 6.9 DisplayConfig Struct Reference

Struct to hold display configuration settings.
`#include <config_types.h>`

### Public Attributes

- std::atomic< int > dispResW

  *dispResW - the width of the display resolution.*
- std::atomic< int > dispResH

  *dispResH - the height of the display resolution.*
- std::atomic< double > fps

### 6.9.1 Detailed Description

Struct to hold display configuration settings.

**Note**

It uses atomic types for thread-safe access to these values.

This struct is used in the Config class to define display configuration settings.

**See also**

dispResW - the width of the display resolution.

dispResH - the height of the display resolution.

fps - the frames per second of the display.

Definition at line 59 of file config_types.h.

### 6.9.2 Member Data Documentation

#### 6.9.2.1 dispResH

`std::atomic<int> DisplayConfig::dispResH`
dispResH - the height of the display resolution.
Definition at line 63 of file config_types.h.

#### 6.9.2.2 dispResW

`std::atomic<int> DisplayConfig::dispResW`
dispResW - the width of the display resolution.
Definition at line 61 of file config_types.h.

#### 6.9.2.3 fps

`std::atomic<double> DisplayConfig::fps`
Definition at line 65 of file config_types.h.
The documentation for this struct was generated from the following file:

- include/config_types.h

## 6.10 GUI Class Reference

Class to manage the GUI components and settings.
`#include <gui.h>`

### Public Member Functions

- GUI ()

  *Constructor for the GUI class. It creates all the GUI components, initializes settings, and sets up connections.*
- bool onExit ()

  *Method that handles the application's exit event.*

### 6.10.1 Detailed Description

Class to manage the GUI components and settings.

**Note**

It uses Qt for creating the GUI and the SettingsDB class for managing settings.

Definition at line 21 of file gui.h.

### 6.10.2 Constructor & Destructor Documentation

#### 6.10.2.1 GUI()

`GUI::GUI ( )`
Constructor for the GUI class. It creates all the GUI components, initializes settings, and sets up connections.
Definition at line 101 of file gui.cpp.

```
00101         {
00102     int argc = 0;
00103     applicationStart = false;
00104 
00105     QApplication app(argc,nullptr);
00106 
00107     QWidget window;
00108     window.setWindowTitle("Interactive Audio Visualizer");
00109 
00110     QVBoxLayout mainLayout;
```

```
00111
00112     initializeComponents();
00113
00114     initializeTexts();
00115
00116     // Audio Settings
00117     QGroupBox audioSettings("Audio Settings");
00118     QVBoxLayout audioLayout;
00119     deviceComboBox->addItems(audioDevices);
00120     audioLayout.addWidget(audioDeviceLabel);
00121     audioLayout.addWidget(deviceComboBox);
00122
00123     // Sample Rate ComboBox
00124     updateSampleRates(deviceComboBox->currentText());
00125     audioLayout.addWidget(sampleRateLabel);
00126     audioLayout.addWidget(sampleRateComboBox);
00127
00128     QObject::connect(deviceComboBox, &QComboBox::currentTextChanged,
00129     [this](const QString &text){
00130         updateSampleRates(text);
00131         updateNumChannelsInfo(text);
00132     });
00133
00134     audioLayout.addWidget(createDropDownList(bufferSizeComboBox,bufferSizeLabel, {"32", "64", "128",
      "256", "512", "1024", "2048", "4096"}));
00135     audioLayout.addWidget(createDropDownList(quantizationComboBox,quantizationLabel, {QString::number(
      AudioHardware::quantizationRatio )} ));
00136     audioLayout.addLayout(numChannelsLayout);
00137     audioSettings.setLayout(&audioLayout);
00138     mainLayout.addWidget(&audioSettings);
00139
00140     // Camera Settings
00141     QGroupBox cameraSettings("Camera Settings");
00142     QVBoxLayout cameraLayout;
00143
00144     // Camera Device ComboBox
00145     cameraDeviceComboBox->addItems(cameraDevices);
00146     cameraLayout.addWidget(cameraDeviceLabel);
00147     cameraLayout.addWidget(cameraDeviceComboBox);
00148
00149     // Resolution ComboBox
00150     updateResolution(cameraDeviceComboBox->currentText());
00151     cameraLayout.addWidget(cameraResolutionLabel);
00152     cameraLayout.addWidget(resolutionComboBox);
00153
00154     QObject::connect(cameraDeviceComboBox, &QComboBox::currentTextChanged,
00155         [this](const QString &text){
00156             updateResolution(text);
00157         });
00158
00159     cameraLayout.addWidget(createDropDownList(frameRateComboBox,cameraFrameRateLabel, {"Auto"}));
00160     cameraSettings.setLayout(&cameraLayout);
00161     mainLayout.addWidget(&cameraSettings);
00162
00163     // Display Settings
00164     QGroupBox displaySettings("Display Settings");
00165     QVBoxLayout displayLayout;
00166     displayLayout.addWidget(createDropDownList(displayResolutionComboBox,screenResolutionLabel,
      displayResolutions));
00167     displayLayout.addWidget(createDropDownList(displayFrameRateComboBox,screenFrameRateLabel,
      {"Auto"}));
00168     displaySettings.setLayout(&displayLayout);
00169     mainLayout.addWidget(&displaySettings);
00170
00171     // IAV Settings
00172     QGroupBox iavSettings("IAV Settings");
00173     QVBoxLayout iavLayout;
00174     iavLayout.addWidget(createDropDownList(frequencyRangeComboBox,iavFrequencyRangeLabel, {"Narrow",
      "Normal", "Wide"}));
00175     iavLayout.addWidget(createDropDownList(roiComboBox,iavRegionOfInterestLabel,
      {"Small","Medium","Large"}));
00176     iavLayout.addWidget(createDropDownList(triggerComboBox,iavTriggerLabel, {"Auto"})); //"Manual",
00177     iavLayout.addWidget(createDropDownList(trackingAlgorithmComboBox,iavTrackingAlgLabel, {"CSRT",
      "KCF"}));
00178     iavLayout.addWidget(createSkipFramesSlider(accuracyLabel, cpuLoadLabel));
00179     iavSettings.setLayout(&iavLayout);
00180     mainLayout.addWidget(&iavSettings);
00181
00182     addExplanations();
00183
00184     // Initialize errorLabel
00185     errorLabel = new QLabel();
00186     errorLabel->setStyleSheet("color:  red;");
00187     errorLabel->setText("Camera resolution cannot exceed display resolution.");
00188     errorLabel->hide(); // Initially hidden
00189     mainLayout.addWidget(errorLabel);
00190
```

```
00191     QPushButton startButton("Start");
00192         QObject::connect(&startButton, &QPushButton::clicked, [this]() {
00193             if(checkResolutionCompatibility()) {
00194                 errorLabel->hide(); // Hide if resolutions are compatible
00195                 saveCurrentStates();
00196                 applicationStart = true;
00197                 QApplication::quit();
00198                 // exiting to start_iav();
00199             } else {
00200                 errorLabel->show(); // Show error message if not compatible
00201             }
00202         });
00203
00204     mainLayout.addWidget(&startButton);
00205
00206     loadCurrentStates();
00207
00208     window.setLayout(&mainLayout);
00209     window.show();
00210
00211     QApplication::exec();
00212 }
```

### 6.10.3 Member Function Documentation

#### 6.10.3.1 onExit()

`bool GUI::onExit ( )`

Method that handles the application's exit event.

**Returns**

bool - true if the application should exit.

Definition at line 374 of file gui.cpp.

```
00374                 {
00375     return !applicationStart;
00376 }
```

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

- include/gui.h
- src/gui.cpp

## 6.11 IAV Class Reference

Class to manage the IAV multi-threaded processing pipeline.

`#include <iav.h>`

### Public Member Functions

- IAV ()

  *Default constructor.*
- ~IAV ()

  *Class destructor.*
- void start ()

  *Starts the IAV processing pipeline.*
- IAV (const IAV &)=delete

  *Copy constructor is deleted to prevent accidental use.*
- IAV (IAV &&)=delete

  *Move constructor is deleted to prevent accidental use.*
- IAV & operator= (const IAV &)=delete

  *Copy assignment operator is deleted to prevent accidental use.*
- IAV && operator= (IAV &&)=delete

  *Move assignment operator is deleted to prevent accidental use.*

### 6.11.1 Detailed Description

Class to manage the IAV multi-threaded processing pipeline.

**See also**

AudioServer class for the audio server operations.

AudioStream class for audio processing and streaming.

Visualizer class for visualizing audio data.

Audiolizer class for converting visual stimulus into audio related data.

Config class for managing configuration settings.

Definition at line 19 of file iav.h.

### 6.11.2 Constructor & Destructor Documentation

#### 6.11.2.1 IAV() [1/3]

```
IAV::IAV ( )
```

Default constructor.

Definition at line 3 of file iav.cpp.

```
00004 {
00005     audioServer.setup_server();
00006 
00007 
      audiolizer.setAudioUpdater(std::bind(&AudioStream::update,&audioStream,std::placeholders::_1,std::placeholders::_2));
00008     visualizer.setAudiolizerUpdater(std::bind(&Audiolizer::turn_Image_into_Sound, &audiolizer,
      std::placeholders::_1,std::placeholders::_2,std::placeholders::_3,std::placeholders::_4));
00009     audioStream.setVisualizerUpdater(std::bind(&Visualizer::updateAudioSignal, &visualizer,
      std::placeholders::_1));
00010 
00011     audServerThread = std::thread (&AudioServer::start_server,&audioServer,std::ref(mtxServer),
      std::ref(cvServer), std::ref(serverStarted));
00012     audioThread = std::thread (&AudioStream::clientConnect,&audioStream,std::ref(mtxServer),
      std::ref(cvServer), std::ref(serverStarted));
00013     visualThread = std::thread(&Visualizer::broadcast,&visualizer);
00014 
00015     cfg.display();
00016 }
```

#### 6.11.2.2 ~IAV()

```
IAV::~IAV ( )
```

Class destructor.

Definition at line 18 of file iav.cpp.

```
00018         {
00019     audioServer.~AudioServer();
00020     audioStream.~AudioStream();
00021     audiolizer.~Audiolizer() ;
00022     visualizer.~Visualizer();
00023 }
```

#### 6.11.2.3 IAV() [2/3]

```
IAV::IAV (
            const IAV &  )  [delete]
```

Copy constructor is deleted to prevent accidental use.

#### 6.11.2.4 IAV() [3/3]

```
IAV::IAV (
            IAV &&  )  [delete]
```

Move constructor is deleted to prevent accidental use.

### 6.11.3 Member Function Documentation

#### 6.11.3.1 operator=() [1/2]

```
IAV & IAV::operator= (
            const IAV &  )  [delete]
```

Copy assignment operator is deleted to prevent accidental use.

#### 6.11.3.2 operator=() [2/2]

```
IAV && IAV::operator= (
            IAV &&  )  [delete]
```

Move assignment operator is deleted to prevent accidental use.

#### 6.11.3.3 start()

```
void IAV::start ( )
```

Starts the IAV processing pipeline.

**Returns**

void

Definition at line 25 of file iav.cpp.

```
00025                {
00026
00027     audServerThread.detach();
00028     audioThread.detach();
00029     visualThread.join();
00030
00031 }
```

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

- include/iav.h
- src/iav.cpp

## 6.12 IAVConfig Struct Reference

Struct to hold IAV configuration settings.

`#include <config_types.h>`

### Public Attributes

- int minFrequency

  *minFrequency - the minimum frequency for sound generation.*
- int maxFrequency

  *maxFrequency - the maximum frequency for sound generation.*
- int roiRadius
- std::string trigger

  *trigger - the type of trigger used for tracking.*
- std::string trackingAlg

  *trackingAlg - the algorithm used for tracking.*

### 6.12.1 Detailed Description

Struct to hold IAV configuration settings.

**Note**

This struct is used in the Config class to define some custom configuration settings based on IAV application needs.

**See also**

minFrequency - the minimum frequency for sound generation.

maxFrequency - the maximum frequency for sound generation.

roiRadius - the radius of the region that is used for tracking. It defines the size of it.

**Warning**

trigger [currently unsupported] - the type of trigger used for tracking.

**See also**

trackingAlg - the algorithm used for tracking.

**Warning**

skipFramesRatio [currently unsupported] - the ratio of frames to skip during tracking.

Definition at line 78 of file config_types.h.

### 6.12.2 Member Data Documentation

#### 6.12.2.1 maxFrequency

```
int IAVConfig::maxFrequency
```

maxFrequency - the maximum frequency for sound generation.
Definition at line 82 of file config_types.h.

#### 6.12.2.2 minFrequency

```
int IAVConfig::minFrequency
```

minFrequency - the minimum frequency for sound generation.
Definition at line 80 of file config_types.h.

#### 6.12.2.3 roiRadius

```
int IAVConfig::roiRadius
```

Definition at line 84 of file config_types.h.

#### 6.12.2.4 trackingAlg

```
std::string IAVConfig::trackingAlg
```

trackingAlg - the algorithm used for tracking.
Definition at line 89 of file config_types.h.

#### 6.12.2.5 trigger

`std::string IAVConfig::trigger`
trigger - the type of trigger used for tracking.

**Warning**

CURRENTLY UNSUPPORTED.

Definition at line 87 of file config_types.h.
The documentation for this struct was generated from the following file:

- include/config_types.h

## 6.13 AudioHardware::Info Struct Reference

Structure representing audio hardware information.
`#include <audiohw.h>`

### Public Attributes

- std::pair< std::string, std::string > card_info
- std::pair< unsigned int, unsigned int > sample_rate_range
- unsigned int numberOfChannels

### 6.13.1 Detailed Description

Structure representing audio hardware information.

**Note**

This structure encapsulates the information about audio hardware devices and their supported features.

**See also**

std::pair<std::string, std::string> card_info - audio card index and audio mixer index, coupled.

sample_rate_range - supportedRates for the list of supported sample rates.

numberOfChannels - the number of output audio device channels

Definition at line 38 of file audiohw.h.

### 6.13.2 Member Data Documentation

#### 6.13.2.1 card_info

`std::pair<std::string, std::string> AudioHardware::Info::card_info`
Definition at line 39 of file audiohw.h.

#### 6.13.2.2 numberOfChannels

`unsigned int AudioHardware::Info::numberOfChannels`
Definition at line 41 of file audiohw.h.

#### 6.13.2.3 sample_rate_range

```
std::pair<unsigned int, unsigned int> AudioHardware::Info::sample_rate_range
```

Definition at line 40 of file audiohw.h.
The documentation for this struct was generated from the following file:

- include/gui/audiohw.h

## 6.14 RegionOfInterest Struct Reference

Struct to hold the region of interest (ROI) data.
`#include <roi.h>`

### Public Attributes

- std::atomic< int > centerX {0}
- std::atomic< int > centerY {0}
- std::atomic< int > volumeW {0}
- std::atomic< int > volumeH {0}

### 6.14.1 Detailed Description

Struct to hold the region of interest (ROI) data.
Contains atomic variables for centerX, centerY, volumeW, and volumeH that allow the efficient and concurrent access and modification of the ROI data. This struct is used to store and manipulate the image image frame used for tracking in the videoTracker and Visualizer classes.
Definition at line 13 of file roi.h.

### 6.14.2 Member Data Documentation

#### 6.14.2.1 centerX

```
std::atomic<int> RegionOfInterest::centerX {0}
```

the x coordinate of the center of the box
Definition at line 15 of file roi.h.

#### 6.14.2.2 centerY

```
std::atomic<int> RegionOfInterest::centerY {0}
```

the y coordinate of the center of the box
Definition at line 17 of file roi.h.

#### 6.14.2.3 volumeH

```
std::atomic<int> RegionOfInterest::volumeH {0}
```

the height of the box
Definition at line 21 of file roi.h.

#### 6.14.2.4 volumeW

```
std::atomic<int> RegionOfInterest::volumeW {0}
```

the width of the box
Definition at line 19 of file roi.h.
The documentation for this struct was generated from the following file:

- include/roi.h

## 6.15 SettingsDB Class Reference

Class to manage settings using a SQLite database.
`#include <settings.h>`

### Public Member Functions

- SettingsDB (const std::string &db_path=Paths::databasePath)

  *Constructor to initialize the database connection.*
- bool saveSettings (const std::unordered_map< std::string, std::string > &settings)

  *Function to save the given settings to the database.*
- std::unordered_map< std::string, std::string > loadSettings ()

  *Function to load the settings from the database.*
- ~SettingsDB ()

  *Destructor to close the database connection and deallocate sql resources.*
- SettingsDB (const SettingsDB &)=delete

  *Copy constructor is deleted to prevent accidental use.*
- SettingsDB & operator= (const SettingsDB &)=delete

  *Copy assignment operator is deleted to prevent accidental use.*
- SettingsDB (SettingsDB &&)=delete

  *Move constructor is deleted to prevent accidental use.*
- SettingsDB & operator= (SettingsDB &&)=delete

  *Move assignment operator is deleted to prevent accidental use.*

### 6.15.1 Detailed Description

Class to manage settings using a SQLite database.
Definition at line 11 of file settings.h.

### 6.15.2 Constructor & Destructor Documentation

#### 6.15.2.1 SettingsDB() [1/3]

```
SettingsDB::SettingsDB (
            const std::string & db_path = Paths::databasePath )  [explicit]
```

Constructor to initialize the database connection.

**Parameters**

| | |
|---|---|
| *db_path* | - The path to the SQLite database file. |

**Note**

If the database file does not exist, it will be created.

Definition at line 5 of file settings.cpp.

```
00005                                           :  dbPath(db_path) {
00006
00007     if (sqlite3_open(dbPath.c_str(), &db) != SQLITE_OK) {
00008         db = nullptr;
00009         return;
00010     }
00011
00012     // Create table if it doesn't exist
```

```
00013     const char* createTableSQL =
00014         "CREATE TABLE IF NOT EXISTS settings ("
00015         "setting_name TEXT PRIMARY KEY,"
00016         "setting_value TEXT NOT NULL);";
00017
00018     char* errMsg = nullptr;
00019     if (sqlite3_exec(db, createTableSQL, nullptr, nullptr, &errMsg) != SQLITE_OK) {
00020         sqlite3_free(errMsg);
00021         sqlite3_close(db);
00022         db = nullptr;
00023     }
00024 }
```

#### 6.15.2.2 ~SettingsDB()

`SettingsDB::~SettingsDB ( )`

Destructor to close the database connection and deallocate sql resources.

Definition at line 26 of file settings.cpp.

```
00026                       {
00027
00028     if (db) {
00029         sqlite3_close(db);
00030     }
00031
00032 }
```

#### 6.15.2.3 SettingsDB() [2/3]

`SettingsDB::SettingsDB (`
`const SettingsDB & ) [delete]`

Copy constructor is deleted to prevent accidental use.

#### 6.15.2.4 SettingsDB() [3/3]

`SettingsDB::SettingsDB (`
`SettingsDB && ) [delete]`

Move constructor is deleted to prevent accidental use.

### 6.15.3 Member Function Documentation

#### 6.15.3.1 loadSettings()

`std::unordered_map< std::string, std::string > SettingsDB::loadSettings ( )`

Function to load the settings from the database.

**Returns**

std::unordered_map<std::string, std::string> - The loaded settings.

Definition at line 80 of file settings.cpp.

```
00080                                                       {
00081     std::unordered_map<std::string, std::string> settings;
00082
00083     if (!db) return settings;
00084
00085     const char* selectSQL = "SELECT setting_name, setting_value FROM settings;";
00086     sqlite3_stmt* stmt;
00087
00088     if (sqlite3_prepare_v2(db, selectSQL, -1, &stmt, nullptr) != SQLITE_OK) {
00089         return settings;
00090     }
00091
00092     while (sqlite3_step(stmt) == SQLITE_ROW) {
00093         std::string name = reinterpret_cast<const char*>(sqlite3_column_text(stmt, 0));
00094         std::string value = reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1));
00095         settings[name] = value;
00096     }
```

```
00097
00098     sqlite3_finalize(stmt);
00099     return settings;
00100 }
```

#### 6.15.3.2 operator=() [1/2]

```
SettingsDB & SettingsDB::operator= (
            const SettingsDB &  )  [delete]
```

Copy assignment operator is deleted to prevent accidental use.

#### 6.15.3.3 operator=() [2/2]

```
SettingsDB & SettingsDB::operator= (
            SettingsDB &&  )  [delete]
```

Move assignment operator is deleted to prevent accidental use.

#### 6.15.3.4 saveSettings()

```
bool SettingsDB::saveSettings (
            const std::unordered_map< std::string, std::string > & settings )
```

Function to save the given settings to the database.

**Parameters**

| | |
|---|---|
| *const* | std::unordered_map<std::string, std::string>& settings - The map of settings to be saved. |

**Returns**

true if the settings were successfully saved, false otherwise.

Definition at line 34 of file settings.cpp.

```
00034                                                                    {
00035     if (!db) return false;
00036
00037     // Begin transaction for better performance
00038     char* errMsg = nullptr;
00039     if (sqlite3_exec(db, "BEGIN TRANSACTION", nullptr, nullptr, &errMsg) != SQLITE_OK) {
00040         sqlite3_free(errMsg);
00041         return false;
00042     }
00043
00044     // First, clear existing settings
00045     const char* clearSQL = "DELETE FROM settings;";
00046     if (sqlite3_exec(db, clearSQL, nullptr, nullptr, &errMsg) != SQLITE_OK) {
00047         sqlite3_free(errMsg);
00048         return false;
00049     }
00050
00051     // Prepare the insert statement
00052     sqlite3_stmt* stmt;
00053     const char* insertSQL = "INSERT INTO settings (setting_name, setting_value) VALUES (?, ?);";
00054     if (sqlite3_prepare_v2(db, insertSQL, -1, &stmt, nullptr) != SQLITE_OK) {
00055         return false;
00056     }
00057
00058     // Insert all settings
00059     for (const auto& [key, value] :  settings) {
00060         sqlite3_bind_text(stmt, 1, key.c_str(), -1, SQLITE_STATIC);
00061         sqlite3_bind_text(stmt, 2, value.c_str(), -1, SQLITE_STATIC);
00062
00063         if (sqlite3_step(stmt) != SQLITE_DONE) {
00064             sqlite3_finalize(stmt);
00065             return false;
00066         }
00067         sqlite3_reset(stmt);
00068     }
00069
00070     sqlite3_finalize(stmt);
00071
```

```
00072     if (sqlite3_exec(db, "COMMIT", nullptr, nullptr, &errMsg) != SQLITE_OK) {
00073         sqlite3_free(errMsg);
00074         return false;
00075     }
00076
00077     return true;
00078 }
```

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

- include/settings.h
- src/settings.cpp

## 6.16 Sine Class Reference

Class responsible for generating sine wave signals for audio processing.
`#include <sine.h>`

### Public Member Functions

- Sine ()

  *Default constructor.*
- void setVisualizerUpdater (std::function< void(float)>)

  *Function to set the visualizer update callback.*
- void setMonoSignal (Tone &, float *[2])

  *Function to generate a mono sine wave signal for a given tone.*
- void setStereoSignal (Tone &, float *[2])

  *Function to generate a stereo sine wave signal for a given tone.*

### 6.16.1 Detailed Description

Class responsible for generating sine wave signals for audio processing.
Definition at line 12 of file sine.h.

### 6.16.2 Constructor & Destructor Documentation

#### 6.16.2.1 Sine()

```
Sine::Sine ( )
```

Default constructor.
Definition at line 10 of file sine.cpp.

```
00010           :audiocfg(Config::getInstance().audconf){
00011     rads_per_sample = 0.;
00012     prevfreq=0;
00013     phase=0.0;
00014 }
```

### 6.16.3 Member Function Documentation

#### 6.16.3.1 setMonoSignal()

```
void Sine::setMonoSignal (
            Tone & tone,
            float * monoBuffer[2] )
```

Function to generate a mono sine wave signal for a given tone.

**Parameters**

| | |
|---|---|
| *tone* | - The tone for which the sine wave signal should be generated. |
| *buffer* | - The buffer to store the generated sine wave signal. |

**Note**

The buffer size received has two channels (to allow compatibillity with stereo), but the the first channel is only used for writting.

Definition at line 20 of file sine.cpp.

```
00020                                                    {
00021
00022     int frequency = tone.frequency.load();
00023     float amplitude = tone.volume.load();
00024
00025     if (frequency != prevfreq){ // reduce number of calculations
00026         rads_per_sample = (static_cast<float>(frequency * 2.* M_PI)) /
      static_cast<float>(audiocfg.sampleRate.load()); //radians traspotition per time unit
00027         prevfreq = frequency;
00028     }
00029
00030     for (int i=0;i<audiocfg.bufferSize.load();i++){
00031         float value = amplitude*(float)sin(phase);
00032         monoBuffer[0][i] = value;
00033         phase+=rads_per_sample;                 // shift phase by amount of rads_per_sample
00034         if (phase >= 2*M_PI) phase=0;           // if phase reaches 2pi , zero it down.
00035
00036         updateVisualizer(value); // fill the shareable ring buffer
00037     }
00038 }
```

### 6.16.3.2 setStereoSignal()

```
void Sine::setStereoSignal (
            Tone & tone,
            float * stereoBuffer[2] )
```

Function to generate a stereo sine wave signal for a given tone.

**Parameters**

| | |
|---|---|
| *tone* | - The tone for which the stereo sine wave signal should be generated. |
| *buffer* | - The buffer to store the generated stereo sine wave signal. |

Definition at line 40 of file sine.cpp.

```
00040                                                          {
00041
00042     int frequency = tone.frequency.load();
00043     float amplitude = tone.volume.load();
00044
00045     if (frequency != prevfreq){ // reduce number of calculations
00046         rads_per_sample = (static_cast<float>(frequency * 2.  * M_PI)) /
      static_cast<float>(audiocfg.sampleRate.load()); //radians traspotition per time unit
00047         prevfreq = frequency;
00048     }
00049
00050     for (int i=0;i<audiocfg.bufferSize.load();i++){
00051         float value = amplitude*(float)sin(phase);
00052         stereoBuffer[0][i] = value;
00053         stereoBuffer[1][i] = value;
00054         phase+=rads_per_sample;                 // shift phase by amount of rads_per_sample
00055         if (phase >= 2*M_PI) phase=0;           // if phase reaches 2pi , zero it down.
00056
00057         updateVisualizer(value); // fill the shareable ring buffer
00058     }
00059 }
```

### 6.16.3.3 setVisualizerUpdater()

```
void Sine::setVisualizerUpdater (
```

```
std::function< void(float)> updater )
```

Function to set the visualizer update callback.

**Parameters**

| | |
|---|---|
| *callback* | - The function to be called when a new visualization update is required. |

**Returns**

void

Definition at line 16 of file sine.cpp.

```
00016                                                                    {
00017     updateVisualizer = std::move(updater);
00018 }
```

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

- include/sine.h
- src/sine.cpp

## 6.17 Spectrogram Class Reference

Ring buffer class to generate a spectrogram of the audio signal using the Fast Fourier Transform (FFT).

`#include <spectrogram.h>`

### Public Member Functions

- Spectrogram ()

  *Default constructor.*
- ~Spectrogram ()

  *Class destructor.*
- Spectrogram (const Spectrogram &)=delete

  *Copy constructor is deleted to prevent accidental use.*
- Spectrogram (Spectrogram &&)=delete

  *Move constructor is deleted to prevent accidental use.*
- Spectrogram & operator= (const Spectrogram &)=delete

  *Copy assignment operator is deleted to prevent accidental use.*
- Spectrogram & operator= (Spectrogram &&)=delete

  *Move assignment operator is deleted to prevent accidental use.*
- int get_numAudioSamples ()

  *Returns the number of audio samples used to calculate the spectrogram.*
- int get_numFFTPoints ()

  *Returns the number of FFT points.*
- bool write (const float &)

  *Function used to set the ring buffer with a value to the write position of the FIFO.*
- bool readBatch (std::vector< float > &, float &, float &)

  *Function used to read a batch of audio samples from the FIFO and store them in a provided vector.*

### 6.17.1 Detailed Description

Ring buffer class to generate a spectrogram of the audio signal using the Fast Fourier Transform (FFT).

**Note**

The spectrogram is calculated using the FFTW3 library.

Definition at line 15 of file spectrogram.h.

## 6.17.2 Constructor & Destructor Documentation

### 6.17.2.1 Spectrogram() [1/3]

`Spectrogram::Spectrogram ( )`

Default constructor.

Definition at line 5 of file spectrogram.cpp.

```
00005                         :  readpos(0), writepos(0), fft_in(700){
00006
00007   calculateNFFT();
00008
00009   initialize_hamming(numAudioSamples);
00010
00011   auto fifoSize {numAudioSamples*2};
00012   ringBuffer.reserve(fifoSize);
00013   ringBuffer.resize(fifoSize);
00014
00015   // fft_in = static_cast<fftw_complex*> (fftw_malloc(sizeof(fftw_complex) * numAudioSamples));
00016   fft_in.reserve(numAudioSamples);
00017   fft_in.resize(numAudioSamples);
00018   fft_out = static_cast<fftw_complex*> (fftw_malloc(sizeof(fftw_complex) * numFFTPoints));
00019
00020   plan = fftw_plan_dft_r2c_1d(numAudioSamples, fft_in.data(), fft_out, FFTW_ESTIMATE); //FFTW_MEASURE
00021
00022   minMagnitude = INT_MAX, maxMagnitude = 0.;
00023
00024 }
```

### 6.17.2.2 ~Spectrogram()

`Spectrogram::~Spectrogram ( )`

Class destructor.

Definition at line 42 of file spectrogram.cpp.

```
00042                           {
00043   if (plan) fftw_destroy_plan(plan);
00044   fftw_cleanup();
00045   // fftw_free(fft_in);
00046   fftw_free(fft_out);
00047 }
```

### 6.17.2.3 Spectrogram() [2/3]

`Spectrogram::Spectrogram ( const Spectrogram & ) [delete]`

Copy constructor is deleted to prevent accidental use.

### 6.17.2.4 Spectrogram() [3/3]

`Spectrogram::Spectrogram ( Spectrogram && ) [delete]`

Move constructor is deleted to prevent accidental use.

## 6.17.3 Member Function Documentation

### 6.17.3.1 get_numAudioSamples()

`int Spectrogram::get_numAudioSamples ( )`

Returns the number of audio samples used to calculate the spectrogram.

**Note**

the number of audio samples defines the capacity of the ring buffer (FIFO).

**Returns**

void

Definition at line 49 of file spectrogram.cpp.

```
00049                                       {
00050   return numAudioSamples;
00051 }
```

#### 6.17.3.2 get_numFFTPoints()

```
int Spectrogram::get_numFFTPoints ( )
```

Returns the number of FFT points.

**Returns**

int - The number of FFT points

Definition at line 53 of file spectrogram.cpp.

```
00053                                    {
00054     return numFFTPoints;
00055 }
```

#### 6.17.3.3 operator=() [1/2]

```
Spectrogram & Spectrogram::operator= (
            const Spectrogram &  )  [delete]
```

Copy assignment operator is deleted to prevent accidental use.

#### 6.17.3.4 operator=() [2/2]

```
Spectrogram & Spectrogram::operator= (
            Spectrogram &&  )  [delete]
```

Move assignment operator is deleted to prevent accidental use.

#### 6.17.3.5 readBatch()

```
bool Spectrogram::readBatch (
            std::vector< float > & result,
            float & min_magnitude,
            float & max_magnitude )
```

Function used to read a batch of audio samples from the FIFO and store them in a provided vector.

**Parameters**

| | |
|---|---|
| *std::vector<float>&* | - the vector to store the read audio samples |
| *float&* | - the minimum magnitude of the audio samples in the batch |
| *float&* | - the maximum magnitude of the audio samples in the batch |

**Returns**

bool - Returns true if successful, false otherwise

Definition at line 74 of file spectrogram.cpp.

```
00074                                                                                {
```

```
00075
00076     // Calculate the start index for reading the last N samples in a forward manner
00077     size_t startIndex = (writepos.load() - numAudioSamples + 1 + ringBuffer.size()) %
      ringBuffer.size();
00078
00079     // Read the N samples starting from startIndex
00080     size_t readPos = startIndex;
00081     for (size_t i = 0; i < (size_t)numAudioSamples; ++i) {
00082
00083         // fft_in[i][0]=ringBuffer[readPos] * hamming_window[i];
00084         // fft_in[i][1]=0;
00085         fft_in[i]=ringBuffer[readPos] * hamming_window[i];
00086
00087         readPos = (readPos + 1) % ringBuffer.size();  // Move to the next sample (circular)
00088     }
00089
00090     fftw_execute(plan);
00091
00092     for (int i = 0; i < numFFTPoints; ++i) {
00093         // Magnitude of the complex number
00094         float magnitude = static_cast<float>(std::sqrt(fft_out[i][0] * fft_out[i][0] + fft_out[i][1] *
      fft_out[i][1]));
00095
00096         minMagnitude = std::min(minMagnitude,magnitude);
00097         maxMagnitude = std::max(maxMagnitude,magnitude);
00098
00099         result[i] = magnitude;
00100     }
00101
00102     min_magnitude = minMagnitude;
00103     max_magnitude = maxMagnitude;
00104
00105     // Update readpos to point to the next sample to be read
00106     readpos.store((readPos + 1) % ringBuffer.size());
00107
00108     return true;
00109 }
```

#### 6.17.3.6 write()

```
bool Spectrogram::write (
            const float & arg )
```

Function used to set the ring buffer with a value to the write position of the FIFO.

**Parameters**

| | |
|---|---|
| *const* | float& - the audio sample to be written |

**Returns**

bool - Returns true if successful, false otherwise

Definition at line 57 of file spectrogram.cpp.

```
00057                                   { // T&& arg --> rvalue reference
00058     auto writePos = writepos.load(); //??
00059     auto nextWritePos = (writePos + 1) % ringBuffer.size();
00060
00061     // if the buffer is full, overwrite the oldest data
00062     if (nextWritePos == readpos.load()) {
00063         auto readPos = (readpos.load() + 1) % ringBuffer.size();
00064         readpos.store(readPos, std::memory_order_release);
00065     }
00066
00067     // write data to the buffer
00068     ringBuffer[writePos] = arg;
00069     writepos.store(nextWritePos);
00070     return true;
00071
00072 }
```

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

- include/spectrogram.h
- src/spectrogram.cpp

# 6.18 Timer Class Reference

A class responsible for managing a timer.
`#include <timer.h>`

## Public Member Functions

- Timer ()

  *Default constructor. Initializes the timer with a default time of 0 seconds.*
- void setTimer (int)

  *Sets the timer to a specific time in seconds.*
- void start ()

  *Starts the timer.*
- bool update (int &)

  *Updates the timer.*
- int getRemainingTime () const

  *Gets the remaining time for display or logging purposes.*
- bool isTimerFinished () const

  *Checks if the timer has finished.*

## 6.18.1 Detailed Description

A class responsible for managing a timer.
Definition at line 9 of file timer.h.

## 6.18.2 Constructor & Destructor Documentation

### 6.18.2.1 Timer()

`Timer::Timer ( )`
Default constructor. Initializes the timer with a default time of 0 seconds.
Definition at line 4 of file timer.cpp.
```
00004 :  initialSeconds(0), remainingTimeMilliseconds(0){}
```

## 6.18.3 Member Function Documentation

### 6.18.3.1 getRemainingTime()

`int Timer::getRemainingTime ( ) const`
Gets the remaining time for display or logging purposes.

**Returns**

int - The remaining time in seconds.

Definition at line 39 of file timer.cpp.
```
00039                                {
00040     return remainingTimeMilliseconds / 1000;  // Convert milliseconds back to seconds
00041 }
```

#### 6.18.3.2 isTimerFinished()

```
bool Timer::isTimerFinished ( ) const
```

Checks if the timer has finished.

**Returns**

bool - True if the timer has finished, false otherwise.

Definition at line 44 of file timer.cpp.

```
00044                                   {
00045     return remainingTimeMilliseconds <= 0;
00046 }
```

#### 6.18.3.3 setTimer()

```
void Timer::setTimer (
            int seconds )
```

Sets the timer to a specific time in seconds.

**Parameters**

| | |
|---|---|
| *seconds* | - The new time in seconds for the timer. |

**Returns**

void

Definition at line 6 of file timer.cpp.

```
00006                                {
00007     initialSeconds = seconds;
00008     remainingTimeMilliseconds = seconds * 1000;
00009     lastUpdateTime = std::chrono::steady_clock::now();
00010 }
```

#### 6.18.3.4 start()

```
void Timer::start ( )
```

Starts the timer.

**Returns**

void

Definition at line 12 of file timer.cpp.

```
00012                  {
00013     lastUpdateTime = std::chrono::steady_clock::now();
00014 }
```

#### 6.18.3.5 update()

```
bool Timer::update (
            int & secondsElapsed )
```

Updates the timer.

**Returns**

void

Definition at line 16 of file timer.cpp.

```
00016                                     {
00017
00018     // Calculate time passed since last update
00019     auto now = std::chrono::steady_clock::now();
00020     std::chrono::duration<int64_t, std::milli> elapsed =
      std::chrono::duration_cast<std::chrono::milliseconds>(now - lastUpdateTime);
```

```
00021
00022     // Subtract elapsed milliseconds from the remaining time
00023     remainingTimeMilliseconds -= static_cast<int>(elapsed.count());
00024     lastUpdateTime = now;  // Update last time frame
00025
00026     // If the timer has finished
00027     if (remainingTimeMilliseconds <= 0) {
00028         remainingTimeMilliseconds = 0;
00029         secondsElapsed = initialSeconds;  // Set elapsed time to the initial value
00030         return true;
00031     }
00032
00033     // Return the elapsed time in seconds (rounded)
00034     secondsElapsed = remainingTimeMilliseconds;
00035     return false;
00036 }
```

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

- include/timer.h
- src/timer.cpp

# 6.19 Tone Struct Reference

A structure to represent a tone with its frequency and volume.

`#include <tone.h>`

## Public Attributes

- std::atomic< int > frequency {0}

  *the frequency of the tone in Hz*
- std::atomic< float > volume {0.f}

  *the volume of the tone, ranging from 0.0 to 1.0*

## 6.19.1 Detailed Description

A structure to represent a tone with its frequency and volume.

**Note**

The frequency is in Hz, and the volume is in a range from 0.0 to 1.0.

The default frequency and volume are both set to 0.

**See also**

frequency and volume

Definition at line 12 of file tone.h.

## 6.19.2 Member Data Documentation

### 6.19.2.1 frequency

`std::atomic<int> Tone::frequency {0}`

the frequency of the tone in Hz

Definition at line 14 of file tone.h.

### 6.19.2.2 volume

`std::atomic<float> Tone::volume {0.f}`

the volume of the tone, ranging from 0.0 to 1.0

Definition at line 16 of file tone.h.

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

- include/tone.h

## 6.20 Trigger Class Reference

A class responsible for managing the trigger behavior.
`#include <trigger.h>`

### Public Member Functions

- Trigger ()

  *Default constructor.*
- bool isTrackingEnabled (float &)

  *Function used to enable or disable tracking mode.*
- void reset ()

  *Function to reset the trigger's state to default (set tracking mode to false).*
- Timer * getTimer ()

  *Function to get a pointer to the timer object used for tracking mode.*

### 6.20.1 Detailed Description

A class responsible for managing the trigger behavior.

**Note**

This class handles the enabling and disabling of tracking mode based on a timer.

Definition at line 15 of file trigger.h.

### 6.20.2 Constructor & Destructor Documentation

#### 6.20.2.1 Trigger()

`Trigger::Trigger ( )`
Default constructor.

**Note**

Initializes a timer with a default time duration of 5 seconds.

The trigger is disabled by default.

Definition at line 4 of file trigger.cpp.

```
00004                   :  iavcfg
      (Config::getInstance().iavconf),timeDurationSec(photo_countdown_sec),mode(0)
00005 {
00006
00007     timer.setTimer(timeDurationSec);
00008
00009     // if (iavcfg.trigger == "Auto"){}
00010     // else if (iavcfg.trigger == "Manual"){}
00011 }
```

### 6.20.3 Member Function Documentation

#### 6.20.3.1 getTimer()

`Timer * Trigger::getTimer ( )`
Function to get a pointer to the timer object used for tracking mode.

**Note**

This function is used for testing purposes.

**Returns**

Timer* - A pointer to the timer object.

Definition at line 38 of file trigger.cpp.

```
00038                          {
00039     return &timer;
00040 }
```

#### 6.20.3.2 isTrackingEnabled()

```
bool Trigger::isTrackingEnabled (
            float & remaining_percentage )
```

Function used to enable or disable tracking mode.
The enabling and disabling of tracking mode is handled using a timer. *In the future this function should be replaced with a more efficient mechanism to handle enabling and disabling based on keyboard inputs.

**Note**

The mode is disabled by default.

**Parameters**

| | |
|---|---|
| *float&* | - True to enable tracking mode, false to disable it. |

**Returns**

bool - True if tracking mode is enabled, false otherwise.

Definition at line 18 of file trigger.cpp.

```
00018                                                    {
00019
00020         if (timer.isTimerFinished()){
00021             _modeToggle();
00022             timer.setTimer(timeDurationSec);
00023         }
00024
00025         int millisecondsElapsed;
00026
00027         timer.update(millisecondsElapsed);
00028         remaining_percentage = (static_cast<float> (millisecondsElapsed) /
      static_cast<float>(timeDurationSec*1000));
00029
00030         return mode;
00031 }
```

#### 6.20.3.3 reset()

```
void Trigger::reset ( )
```

Function to reset the trigger's state to default (set tracking mode to false).

**Returns**

void

Definition at line 33 of file trigger.cpp.

```
00033                    {
00034     mode = 0;
00035     timeDurationSec = photo_countdown_sec;
00036 }
```

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

- include/trigger.h
- src/trigger.cpp

## 6.21 VideoTracker Class Reference

A class responsible for tracking objects in the camera feed.
`#include <videotracker.h>`

### Public Member Functions

- VideoTracker ()

  *Default constructor.*
- void initializeTracker (const cv::Mat &)

  *Function to initialize the tracker with a given image.*
- bool trackObject (const cv::Mat &, RegionOfInterest &)

  *Function to track the object in the given image.*

### 6.21.1 Detailed Description

A class responsible for tracking objects in the camera feed.

**Note**

This class uses OpenCV's built-in tracker API for object tracking.

Definition at line 12 of file videotracker.h.

### 6.21.2 Constructor & Destructor Documentation

#### 6.21.2.1 VideoTracker()

```
VideoTracker::VideoTracker ( )
```

Default constructor.
Definition at line 4 of file videotracker.cpp.

```
00004                           {
00005
00006     int radius=cfg.iavconf.roiRadius;
00007     int W=cfg.camconf.camResW.load();
00008     int H=cfg.camconf.camResH.load();
00009     cv::Rect temp((W/2)-radius,(H/2)-radius,radius*2,radius*2);
00010     centerBox=temp;
00011     boundingBox=temp;
00012 }
```

### 6.21.3 Member Function Documentation

#### 6.21.3.1 initializeTracker()

```
void VideoTracker::initializeTracker (
            const cv::Mat & frame )
```

Function to initialize the tracker with a given image.

**Parameters**

| | | |
|---|---|---|
| in | *const* | cv::Mat& - Image containing the object to be tracked. |

**Returns**

void

Definition at line 15 of file videotracker.cpp.

```
00015                                                   {
00016     if (cfg.iavconf.trackingAlg == "CSRT") {
00017         tracker = cv::TrackerCSRT::create();
00018     } else if (cfg.iavconf.trackingAlg == "KCF") {
00019         tracker = cv::TrackerKCF::create();
00020     }
00021     boundingBox = centerBox;
00022     tracker->init(frame, boundingBox);
00023 }
```

#### 6.21.3.2 trackObject()

```
bool VideoTracker::trackObject (
            const cv::Mat & frame,
            RegionOfInterest & trackingSig )
```

Function to track the object in the given image.

**Parameters**

| | | |
|---|---|---|
| in | *const* | cv::Mat& - Image containing the object to be tracked. |
| in | *RegionOfInterest&* | - Region of interest where the object should be tracked. |

**Returns**

bool - True if the object is successfully tracked, false otherwise.

Definition at line 25 of file videotracker.cpp.

```
00025                                                                                   {
00026
00027         bool trackingUpdated = tracker->update(frame, boundingBox);
00028         if (trackingUpdated){
00029
00030     // @ comment :  this gives the center x,y and the w and h
00031             trackingSig.centerX.store(static_cast<int>(boundingBox.x + boundingBox.width/2));
00032             trackingSig.centerY.store(static_cast<int>(boundingBox.y + boundingBox.height/2));
00033             trackingSig.volumeW.store(static_cast<int>(boundingBox.width));
00034             trackingSig.volumeH.store(static_cast<int>(boundingBox.height));
00035     // @ comment :  this gives the TOPLEFT corner x,y and the w and h
00036             // trackingSig.centerX.store(static_cast<int>(boundingBox.x));
00037             // trackingSig.centerY.store(static_cast<int>(boundingBox.y));
00038             // trackingSig.volumeW.store(static_cast<int>(boundingBox.width));
00039             // trackingSig.volumeH.store(static_cast<int>(boundingBox.height));
00040
00041         }
00042         return trackingUpdated;
00043
00044 }
```

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

- include/videotracker.h
- src/videotracker.cpp

## 6.22 Visualizer Class Reference

This class is responsible for managing the camera feed, tracking objects, triggering, and broadcasting the visualized frame to the IAV pipeline.

```
#include <visualizer.h>
```

### Public Member Functions

- Visualizer ()

  *Default constructor.*

- ~Visualizer ()

  *Class destructor.*

- Visualizer (const Visualizer &)=delete

*Copy constructor is deleted to prevent accidental use.*

- Visualizer (Visualizer &&)=delete

  *Move constructor is deleted to prevent accidental use.*

- Visualizer & operator= (const Visualizer &)=delete

  *Copy assignment operator is deleted to prevent accidental use.*

- Visualizer & operator= (Visualizer &&)=delete

  *Move assignment operator is deleted to prevent accidental use.*

- void broadcast ()

  *Broadcasts the visualized frame to the IAV processing pipeline.*

- void setAudiolizerUpdater (std::function< void(const bool, const bool, const RegionOfInterest &, Tone &)>)

  *Sets the function that is to be used for receiving the Audiolizer's update signal.*

- void updateAudioSignal (float)

  *Receives the audio signal by the Sine instance, to update the Waveform and Spectrogram instances.*

## 6.22.1 Detailed Description

This class is responsible for managing the camera feed, tracking objects, triggering, and broadcasting the visualized frame to the IAV pipeline.
Definition at line 17 of file visualizer.h.

## 6.22.2 Constructor & Destructor Documentation

### 6.22.2.1 Visualizer() [1/3]

```
Visualizer::Visualizer ( )
```

Default constructor.
Definition at line 11 of file visualizer.cpp.

```
00011                       {
00012
00013     int W=cfg.dispconf.dispResW.load();
00014     int H=cfg.dispconf.dispResH.load();
00015
00016     cv::namedWindow("Interactive Audio Visualizer",cv::WINDOW_AUTOSIZE);
00017     cv::Mat img( H , W, CV_8UC3,cv::Scalar(0,0,0));
00018     visualFrame = img;
00019
00020     _create_camMask();
00021
00022     int cameraW = cfg.camconf.camResW.load();
00023     int cameraH = cfg.camconf.camResH.load();
00024     LR = W - cameraW;
00025     TB = H - cameraH;
00026
00027     transpose_ratio_x = static_cast<float>(W) / static_cast<float>(cameraW) / 2.0f;
00028     transpose_ratio_y = static_cast<float>(H) / static_cast<float>(cameraH) / 2.0f;
00029
00030     trackingToggle = false;
00031
00032     int nfft = spectrogram.get_numFFTPoints();
00033     specMagnitude.reserve(nfft);
00034     specMagnitude.resize(nfft);
00035
00036     _set_freq_midBoundaries();
00037 }
```

### 6.22.2.2 ~Visualizer()

```
Visualizer::~Visualizer ( )
```

Class destructor.
Definition at line 187 of file visualizer.cpp.

```
00187                        {
00188     cv::destroyWindow("Interactive Audio Visualizer");
00189     visualFrame.release();
00190     cameraFrame.release();
```

```
00191     camBinaryMask.release();
00192 }
```

#### 6.22.2.3 Visualizer() [2/3]

```
Visualizer::Visualizer (
            const Visualizer &  )  [delete]
```

Copy constructor is deleted to prevent accidental use.

#### 6.22.2.4 Visualizer() [3/3]

```
Visualizer::Visualizer (
            Visualizer &&  )  [delete]
```

Move constructor is deleted to prevent accidental use.

### 6.22.3 Member Function Documentation

#### 6.22.3.1 broadcast()

```
void Visualizer::broadcast ( )
```

Broadcasts the visualized frame to the IAV processing pipeline.

**Note**

Runs in a separate thread.

**Returns**

void

Definition at line 142 of file visualizer.cpp.

```
00142                           {
00143 
00144     bool trackingEnabled,trackingUpdated;
00145     RegionOfInterest trackingSig;
00146     Tone tone;
00147 
00148     while(true){
00149 
00150         // camera in visualizer
00151         bool frameElapsed = camera.capture(cameraFrame); // get data
00152         if (!frameElapsed){
00153             return;
00154         }
00155 
00156         float remaining_percentage;
00157         trackingEnabled = trigger.isTrackingEnabled(remaining_percentage);
00158         updateTrackingMode(trackingEnabled);
00159 
00160         if (trackingEnabled){ // preprocess visual_frame -->   doesn't depict the frame, it just edits
      it so it does not require a new frame to be captured by the camera.
00161 
00162             trackingUpdated = videoTracker.trackObject(cameraFrame, trackingSig);
00163 
00164             if (!trackingUpdated){
00165                 // reset
00166                 trigger.reset();
00167             }
00168 
00169             // update the current visualframe according to the changing of the tracking stimulus
00170             _set_BG_manually(tone);
00171             _set_FG_manually(trackingSig);
00172 
00173 
00174         }else{
00175             _setToCamera(remaining_percentage);
00176             trackingUpdated = trackingEnabled = false;
00177         }
00178 
```

```
00179         updateAudioLizer(trackingUpdated, trackingEnabled, trackingSig, tone);
00180
00181         bool exit_msg = _showFrame();
00182         if (exit_msg)
00183             break;
00184     }
00185 }
```

#### 6.22.3.2 operator=() [1/2]

```
Visualizer & Visualizer::operator= (
            const Visualizer &  )  [delete]
```

Copy assignment operator is deleted to prevent accidental use.

#### 6.22.3.3 operator=() [2/2]

```
Visualizer & Visualizer::operator= (
            Visualizer &&  )  [delete]
```

Move assignment operator is deleted to prevent accidental use.

#### 6.22.3.4 setAudiolizerUpdater()

```
void Visualizer::setAudiolizerUpdater (
            std::function< void(const bool, const bool, const RegionOfInterest &, Tone &)>
function )
```

Sets the function that is to be used for receiving the Audiolizer's update signal.

**Parameters**

| | |
|---|---|
| *std::function<void(const* | bool, const bool, const RegionOfInterest&, Tone&)> - The function to be called for receiving the Audiolizer's update signal. |

**Note**

AudiolizerUpdater

**Parameters**

| | |
|---|---|
| *1* | const bool trackingUpdated - variable that indicates whether there is a new tracking signal.. |

**Note**

AudiolizerUpdater

**Parameters**

| | |
|---|---|
| *2* | const bool trackingEnabled - variable that indicates whether the tracking is enabled or not. |

**Note**

AudiolizerUpdater

**Parameters**

| | |
|---|---|
| *3* | const RegionOfInterest& roi - variable passed by reference that updates the value of the current tracking signal. |

**Note**

AudiolizerUpdater

**Parameters**

| | |
|---|---|
| *4* | Tone& - the tone object for storing the current frequency and volume. |

**Returns**

void

Definition at line 39 of file visualizer.cpp.

```
00039
     {
00040     updateAudioLizer = std::move(function);
00041 }
```

#### 6.22.3.5 updateAudioSignal()

```
void Visualizer::updateAudioSignal (
             float newValue )
```

Receives the audio signal by the Sine instance, to update the Waveform and Spectrogram instances.

**Parameters**

| | |
|---|---|
| *float* | - the audio signal received by the Sine instance. |

**Returns**

void

Definition at line 55 of file visualizer.cpp.

```
00055                                                {
00056     waveform.write(newValue);
00057     spectrogram.write(newValue);
00058 }
```

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

- include/visualizer.h
- src/visualizer.cpp

## 6.23 Waveform Class Reference

A circular buffer for storing audio samples.

```
#include <waveform.h>
```

### Public Member Functions

- Waveform ()

  *Default constructor.*

- bool write (const float &)

  *Writes a new sample to the buffer. Advances the write position by one.*

- bool read (float &)

  *Reads a sample from the buffer. Advances the read position by one.*
- bool isEmpty () const

  *Checks if the buffer is empty.*
- bool isFull () const
- size_t size () const

  *Returns the capacity of the buffer.*
- size_t availableForReading () const

  *Returns the number of samples that can be read from the buffer without blocking.*
- void getMinMax (float[2])

  *Calculates and returns the minimum and maximum values in the buffer.*

### 6.23.1 Detailed Description

A circular buffer for storing audio samples.

**Note**

The buffer is circular, meaning that when the write position reaches the buffer size, it wraps around to the beginning.

Definition at line 12 of file waveform.h.

### 6.23.2 Constructor & Destructor Documentation

#### 6.23.2.1 Waveform()

`Waveform::Waveform ( )`

Default constructor.

Definition at line 5 of file waveform.cpp.

```
00005                   :  readpos(0), writepos(0){
00006     // calculate num of buffers per display frame;
00007     int audioSamplesPerFrame = static_cast<int>(cfg.audconf.sampleRate.load() /
      cfg.dispconf.fps.load());
00008     int buffersPerFrame = std::ceil( static_cast<float>(audioSamplesPerFrame) /
      static_cast<float>(cfg.audconf.bufferSize.load()));
00009     audioSamplesPerFrame = (buffersPerFrame+1) * cfg.audconf.bufferSize.load(); // add 1 and make it
      divisible by bufferSize.
00010
00011     waveTable.reserve(audioSamplesPerFrame);
00012     waveTable.resize(audioSamplesPerFrame);
00013
00014     capacity = (size_t)audioSamplesPerFrame;
00015     min = INT_MAX, max = 0.;
00016 }
```

### 6.23.3 Member Function Documentation

#### 6.23.3.1 availableForReading()

`size_t Waveform::availableForReading ( ) const`

Returns the number of samples that can be read from the buffer without blocking.

**Returns**

size_t - the number of samples available for reading.

Definition at line 68 of file waveform.cpp.

```
00068                                           {
00069     size_t writePos = writepos.load();
00070     size_t readPos = readpos.load();
00071     if (writePos >= readPos) {
00072         return writePos - readPos;  // Normal case, no wraparound
00073     } else {
00074         return waveTable.size() - (readPos - writePos);  // Wraparound case
00075     }
00076 }
```

#### 6.23.3.2 getMinMax()

```
void Waveform::getMinMax (
            float minMax[2] )
```

Calculates and returns the minimum and maximum values in the buffer.

**Parameters**

| | | |
|---|---|---|
| out | *float[2]* | min_max - The minimum and maximum values in the buffer. 0th element is minimum, 1st element is maximum. |

**Returns**

void

Retrieves the minimum and maximum values from the waveform.

**Parameters**

| | | |
|---|---|---|
| out | *min* | The minimum value (output parameter). |
| out | *max* | The maximum value (output parameter). |

Definition at line 83 of file waveform.cpp.

```
00083                                                {
00084     minMax[0] = this->min;
00085     minMax[1] = this->max;
00086 }
```

#### 6.23.3.3 isEmpty()

```
bool Waveform::isEmpty ( ) const
```

Checks if the buffer is empty.

**Returns**

bool - true if the buffer is empty, false otherwise.

Definition at line 56 of file waveform.cpp.

```
00056                                {
00057     return readpos.load() == writepos.load();
00058 }
```

#### 6.23.3.4 isFull()

```
bool Waveform::isFull ( ) const
```

Definition at line 60 of file waveform.cpp.

```
00060                               {
00061     return (( writepos.load() + 1 ) % waveTable.size()) == readpos.load();
00062 }
```

#### 6.23.3.5 read()

```
bool Waveform::read (
            float & result )
```

Reads a sample from the buffer. Advances the read position by one.

**Parameters**

| | | |
|---|---|---|
| out | *float&* | sample - the sample to be read. |

**Returns**

bool - true if the sample was successfully read, false otherwise.

Definition at line 38 of file waveform.cpp.

```
00038                                {
00039
00040     auto readPos = readpos.load();
00041
00042     // if buffer is empty return false
00043     if (readPos == writepos.load()){
00044         return false;
00045     }
00046
00047     // move data out of the buffer;
00048     result = waveTable[readPos];
00049
00050     // advance the read pointer
00051     auto nextReadPos = (readPos + 1) % waveTable.size();
00052     readpos.store(nextReadPos);
00053     return true;
00054 }
```

#### 6.23.3.6 size()

```
size_t Waveform::size ( ) const
```

Returns the capacity of the buffer.

**Returns**

size_t - the capacity of the buffer.

Definition at line 64 of file waveform.cpp.

```
00064                               {
00065     return capacity;
00066 }
```

#### 6.23.3.7 write()

```
bool Waveform::write (
            const float & arg )
```

Writes a new sample to the buffer. Advances the write position by one.

**Parameters**

| | | |
|---|---|---|
| in | *float* | sample - the new sample to be written. |

**Returns**

bool - true if the sample was successfully written, false otherwise.

Definition at line 18 of file waveform.cpp.

```
00018                                     { // T&& arg --> rvalue reference
00019     auto writePos = writepos.load(); //??
00020     auto nextWritePos = (writePos + 1) % waveTable.size();
00021
```

```
00022     // if the buffer is full, overwrite the oldest data
00023     if (nextWritePos == readpos.load()) {
00024         auto readPos = (readpos.load() + 1) % waveTable.size();
00025         readpos.store(readPos, std::memory_order_release);
00026     }
00027
00028     min = std::min(min,arg);
00029     max = std::max(max,arg);
00030
00031     // write data to the buffer
00032     waveTable[writePos] = arg;
00033     writepos.store(nextWritePos);
00034     return true;
00035
00036 }
```

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

- include/waveform.h
- src/waveform.cpp

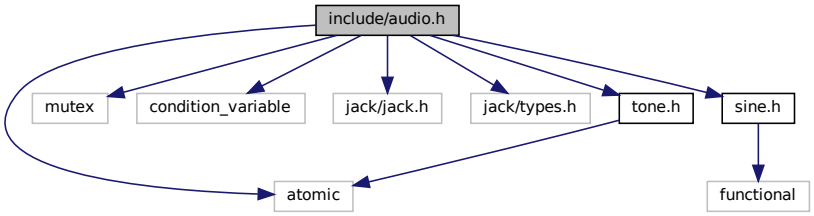
# Chapter 7

# File Documentation

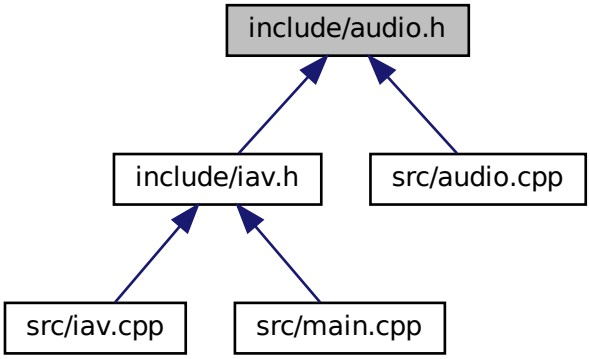
## 7.1 include/audio.h File Reference

```
#include <atomic>
#include <mutex>
#include <condition_variable>
#include <jack/jack.h>
#include <jack/types.h>
#include "sine.h"
#include "tone.h"
```

Include dependency graph for audio.h:



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




## Classes

- class AudioStream

  *A class representing the audio streaming functionality.*

# 7.2 audio.h

Go to the documentation of this file.

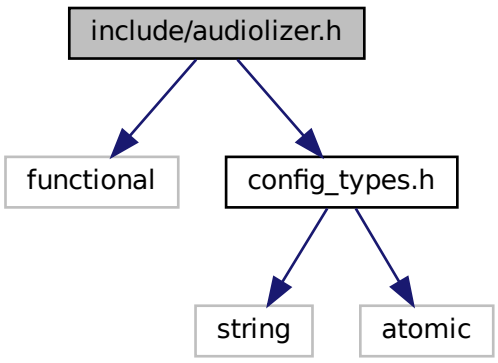
```
00001 #ifndef AUDIO_H
00002 #define AUDIO_H
00003
00004 #include <atomic>
00005 #include <mutex>
00006 #include <condition_variable>
00007 #include <jack/jack.h>
00008 #include <jack/types.h>
00009 #include "sine.h"
00010 #include "tone.h"
00011
00012 class Config;
00013 class Waveform;
00014
00022 class AudioStream{
00023 public:
00024
00028     AudioStream();
00029
00033     ~AudioStream();
00034
00041     void clientConnect(std::mutex&, std::condition_variable&, bool&);
00042
00046     void closeStream();
00047
00052     int streamBuffer();
00053
00060     void update(int,  float);
00061
00068     void setVisualizerUpdater(std::function<void(float)>);
00069
00073      AudioStream(const AudioStream&) = delete;
00074
00078     AudioStream(AudioStream&&)  = delete;
00079
00083     AudioStream& operator=(const AudioStream&) = delete;
00084
00088     AudioStream& operator=(AudioStream&&) = delete;
00089
00090 private:
00091
00092     const AudioConfig& audiocfg;
```

```
00093
00094     const char *client_name ;
00095     const char **todevice;
00096     jack_client_t *client;
00097     jack_port_t * output_ports[2];
00098     float *outputBuffers[2];
00099
00100     Sine sine;
00101     void (Sine::*make_sound)(Tone&,float*[2]) = nullptr;
00102     Tone tone;
00103
00108     static int streamAudio (jack_nframes_t nframes, void *arg);
00109
00110
00113     static void jack_shutdown (void *arg);
00114 };
00115
00116 #endif
00117
00118
00119
00120
00121
```

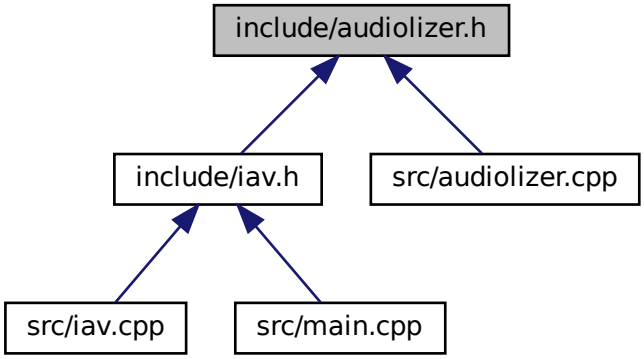
## 7.3 include/audiolizer.h File Reference

```
#include <functional>
#include "config_types.h"
```

Include dependency graph for audiolizer.h:



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




## Classes

- class Audiolizer

  *A class responsible for translating tracking signal into audio frequency.*

# 7.4 audiolizer.h

Go to the documentation of this file.

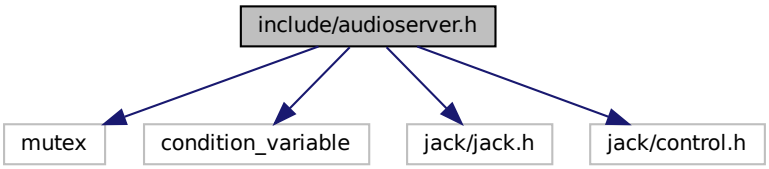
```
00001 #ifndef AUDIOLIZER_H
00002 #define AUDIOLIZER_H
00003
00004 #include <functional>
00005 #include "config_types.h"
00006 struct RegionOfInterest;
00007 class Tone;
00008
00013 class Audiolizer{
00014
00015 public:
00016
00019     Audiolizer();
00020
00028     bool turn_Image_into_Sound(const bool, const bool, const RegionOfInterest&, Tone&);
00029
00036     void setAudioUpdater(std::function<void(int,float)>);
00037
00038 private:
00039
00040     CameraConfig &cameracfg;
00041     IAVConfig &iavcfg;
00042     int frequencyRange,prev_freq;
00043     float volume;
00044
00045     std::function<void(int,float)> updateAudio;
00046
00053     bool translate(const RegionOfInterest&,int&);
00054
00059     void gradualy_fade(int&);
00060
00061     // @TEMPORARY DISABLED
00062     // /*!  @brief Method that is called once during implicit construction to calculate 2 terms (a and
      b).
00063     //  *a and b are latter used for converting frequency range into a logarithic scale
00064     //  *Currently not used.
00065     //  * @param int - the minimum value of the frequency range defined.
00066     //  * @param int - the maximum value of the frequency range defined.
00067     //  * @returns void
00068     //  */
00069     // void init_log_freq_scale();
00070
```

```
00071     // /*!  @brief Method to map the linear values of a predifined frequency range into a logarithmic
      range.
00072     // * Currently not used.
00073     // * @param int& - frequency variable passed by reference.
00074     // */
00075     // void int2log_freq(int&);  // currently not used
00076     // double a,b;
00077 };
00078
00079 #endif
```

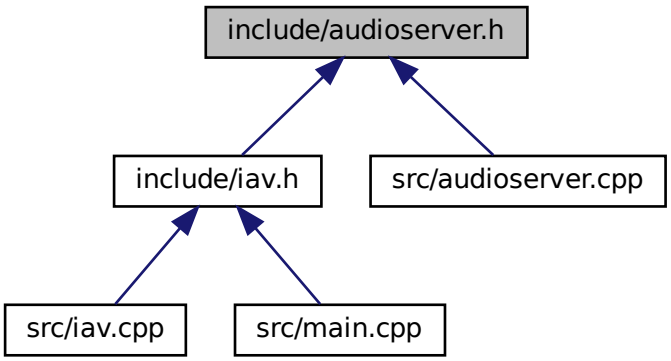
## 7.5 include/audioserver.h File Reference

```
#include <mutex>
#include <condition_variable>
#include <jack/jack.h>
#include <jack/control.h>
```

Include dependency graph for audioserver.h:




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




### Classes

- class AudioServer

  *The jack-audio server running on the alsa drivers.*

### Variables

- const char supported_driver [5] = "alsa"

### 7.5.1 Variable Documentation

#### 7.5.1.1 supported_driver

```
const char supported_driver[5] = "alsa"
```

Definition at line 12 of file audioserver.h.

## 7.6 audioserver.h

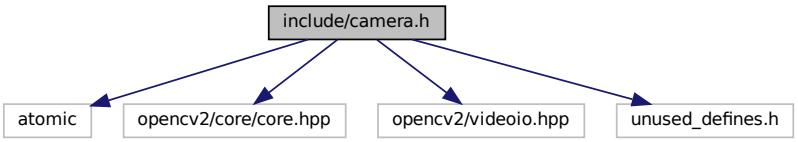
Go to the documentation of this file.

```
00001 #ifndef AUDIOSERVER_H
00002 #define AUDIOSERVER_H
00003 #include <mutex>
00004 #include <condition_variable>
00005 #include <jack/jack.h>
00006 #include <jack/control.h>
00007
00008 // #define SERVER_VERBOSE
00009
00010 struct AudioConfig;
00011
00012 const char supported_driver[5] = "alsa";
00013
00022 class AudioServer{
00023 public:
00026     explicit AudioServer(const char* driverName = supported_driver);
00027
00031     ~AudioServer();
00032
00036     void setup_server();
00037
00044     void start_server(std::mutex&, std::condition_variable&, bool&);
00045
00049     void stop_server();
00050
00051     // Rule of five (5)
00055     AudioServer (const AudioServer&) = delete;
00056
00060     AudioServer (AudioServer&&) = delete;
00061
00065     AudioServer& operator=(const AudioServer&) = delete;
00066
00070     AudioServer& operator=(AudioServer&&) = delete;
00071
00072 private:
00073     jackctl_server_t *server;
00074     const JSList *parameters;
00075     const JSList *drivers;
00076     jackctl_sigmask_t *sigmask;
00077     const char *driver_name;
00078     const AudioConfig& audiocfg;
00079
00083     void change_server_parameters();
00087     void change_ALSAdriver_parameters();
00088
00092     jackctl_driver_t* jackctl_server_get_driver();
00098     static jackctl_parameter_t* jackctl_get_parameter(const JSList*,const char *);
00099
00100 #ifdef SERVER_VERBOSE
00105     static void print_parameters(const JSList*);
00106
00112     static void print_value(union jackctl_parameter_value, jackctl_param_type_t);
00113
00118     void print_driver_info();
00119 #endif
00120
00121 };
00122 #endif
```

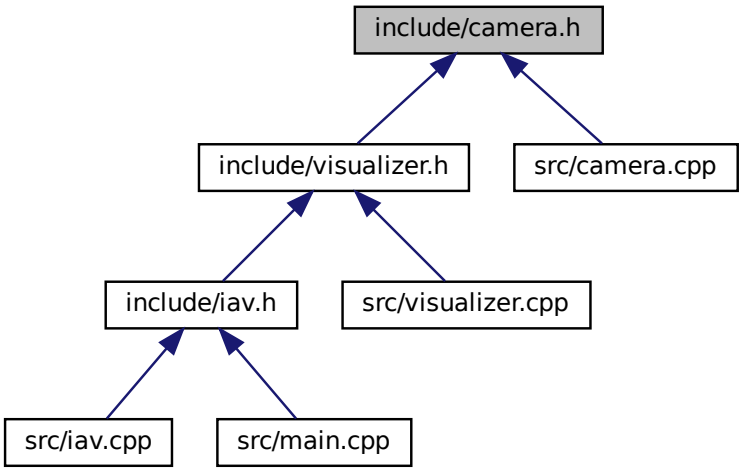
## 7.7 include/camera.h File Reference

```
#include <atomic>
#include <opencv2/core/core.hpp>
#include <opencv2/videoio.hpp>
```

`#include "unused_defines.h"`
Include dependency graph for camera.h:




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




## Classes

- class Camera

  *Class representing a camera object.*

# 7.8 camera.h

Go to the documentation of this file.

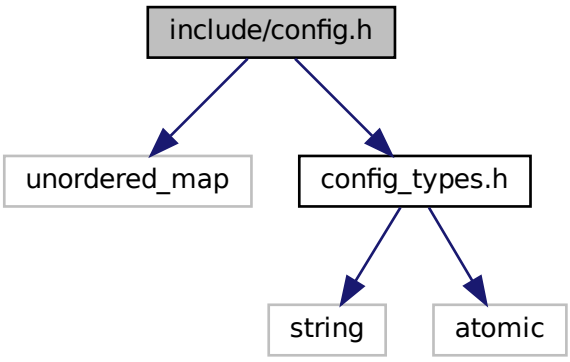
```
00001 #ifndef CAMERA_H
00002 #define CAMERA_H
00003
00004 #include <atomic>
00005 #include <opencv2/core/core.hpp>
00006 #include <opencv2/videoio.hpp>
00007 #include "unused_defines.h"
00008 struct CameraConfig;
00009
00016 class Camera{
00017 public:
00018
00020     Camera();
00021
00023     ~Camera();
00024
00028     bool capture(cv::Mat&);
```

```
00029
00034     bool frame_elapsed();
00035
00039     Camera (const Camera&) = delete;
00040
00044     Camera (Camera&&) = delete;
00045
00049     Camera& operator=(const Camera&) = delete;
00050
00054     Camera& operator=(Camera&&) = delete;
00055
00056 private:
00057
00058     CameraConfig &cameracfg;
00059     // int camW,camH,fps;
00060
00061     std::atomic<bool> frameToggle;
00062     int toggleFrame;
00063     bool atomicChange;
00064
00065     cv::VideoCapture cap;
00066     cv::Mat frame;
00067
00068     void initialize_camera();
00069 };
00070
00071
00072
00073
00074
00075 #endif
```

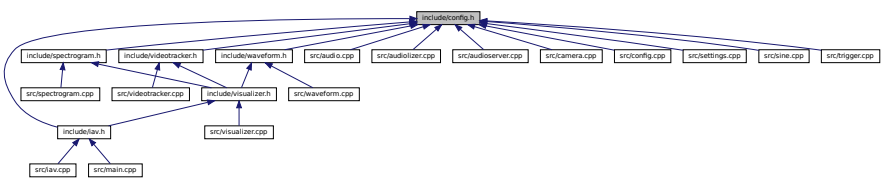
## 7.9 include/config.h File Reference

```
#include <unordered_map>
#include "config_types.h"
```

Include dependency graph for config.h:




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



### Classes

- class Config

  *Singleton class to manage configuration settings, providing a unique point of access to the configuration settings.*

## 7.10 config.h

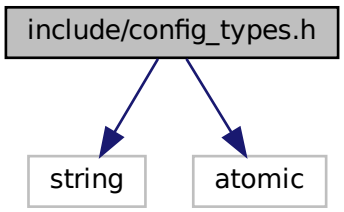
Go to the documentation of this file.

```
00001 #ifndef CONFIG_H
00002 #define CONFIG_H
00003
00004 #include <unordered_map>
00005 #include "config_types.h"
00006
00013 class Config{
00014
00015 public:
00016
00020     static Config& getInstance(){
00021         static Config config;
00022         return config;
00023     }
00024
00028     Config(Config const&)           = delete;
00029
00033     void operator=(Config const&)   = delete;
00034
00038     void display();
00039
00043     AudioConfig audconf;
00044
00048     CameraConfig camconf;
00049
00053     DisplayConfig dispconf;
00054
00058     IAVConfig  iavconf;
00059
00060 private:
00064     Config();
00065
00066     std::unordered_map<std::string, std::string> settings;
00067
00068     bool runAtomicityCheck();
00069
00070 };
00071
00072 #endif
```

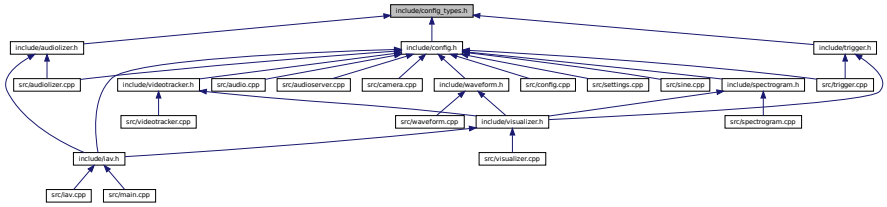
## 7.11 include/config_types.h File Reference

```
#include <string>
#include <atomic>
```

Include dependency graph for config_types.h:



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




## Classes

- struct AudioConfig

  *Struct to hold audio configuration settings.*
- struct CameraConfig

  *Struct to hold camera configuration settings.*
- struct DisplayConfig

  *Struct to hold display configuration settings.*
- struct IAVConfig

  *Struct to hold IAV configuration settings.*

# 7.12 config_types.h

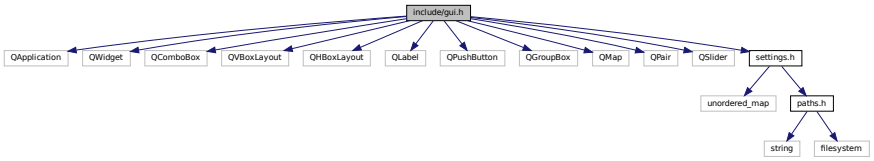
Go to the documentation of this file.

```
00001 #ifndef CONFIG_TYPES_H
00002 #define CONFIG_TYPES_H
00003
00004 #include <string>
00005 #include <atomic>
00006
00018 struct AudioConfig{
00020     std::string audioDevice;
00022     std::atomic<int> sampleRate;
00023     /* @brief quantization - the quantization of the audio data.*/
00024     int quantization;
00025     /* @brief bufferSize - the buffer size for audio data.*/
00026     std::atomic<int> bufferSize;
00027     /* @brief numChannels - the number of output audio channels.*/
00028     std::atomic<unsigned int> numChannels;
00029 };
00030
00040 struct CameraConfig{
00042     std::string device;
00043     /* @brief frameRate - the frame rate of the camera.*/
00044     std::atomic<double> frameRate;
00045     /* @brief camResW - the width of the camera resolution.*/
00046     std::atomic<int> camResW;
00047     /* @brief camResH - the height of the camera resolution.*/
00048     std::atomic<int> camResH;
00049 };
00050
00059 struct DisplayConfig{
00061     std::atomic<int> dispResW;
00063     std::atomic<int> dispResH;
00064     /* @brief fps - the frames per second of the display.*/
00065     std::atomic<double> fps;
00066 };
00067
00078 struct IAVConfig{
00080     int minFrequency;
00082     int maxFrequency;
00083     /* @brief roiRadius - the radius of the region that is used for tracking.  It defines the size of
      it.*/
00084     int roiRadius;
00087     std::string trigger;
00089     std::string trackingAlg;
00090     // int skipFramesRatio;
00091 };
00092
00093
00094 #endif
```

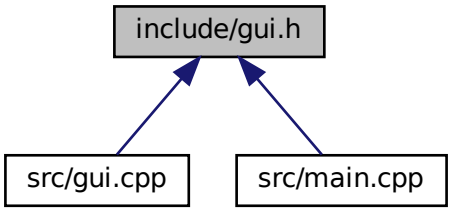
## 7.13 include/gui.h File Reference

```
#include <QApplication>
#include <QWidget>
#include <QComboBox>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QLabel>
#include <QPushButton>
#include <QGroupBox>
#include <QMap>
#include <QPair>
#include <QSlider>
#include "settings.h"
```

Include dependency graph for gui.h:




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




### Classes

- class GUI

  *Class to manage the GUI components and settings.*

## 7.14 gui.h

Go to the documentation of this file.

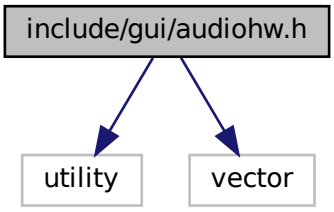
```
00001 #ifndef GUI_H
00002 #define GUI_H
00003
00004 #include <QApplication>
00005 #include <QWidget>
00006 #include <QComboBox>
00007 #include <QVBoxLayout>
00008 #include <QHBoxLayout>
00009 #include <QLabel>
00010 #include <QPushButton>
00011 #include <QGroupBox>
00012 #include <QMap>
```

```
00013 #include <QPair>
00014 #include <QSlider>
00015 #include "settings.h"
00016
00021 class GUI{
00022 public:
00023
00027     GUI();
00028
00032     bool onExit();
00033
00034 private:
00035
00036     bool applicationStart;
00037     double approxFps;
00038     std::vector<std::string> audioExplanations;
00039
00040     void initializeComponents();
00041     void initializeTexts();
00042     void setupComboBoxes(QApplication);
00043
00044     static QWidget* createDropDownList(QComboBox *, QLabel *, const QStringList&);
00045     QWidget* createSkipFramesSlider(QLabel *, QLabel *);
00046
00047     void updateSampleRates(const QString&);
00048     void updateNumChannelsInfo(const QString&);
00049     void updateResolution(const QString&);
00050
00051     void saveCurrentStates();
00052     void loadCurrentStates();
00053     bool checkResolutionCompatibility();
00054
00055     void addExplanations();
00056
00057     SettingsDB settingsDB;
00058
00059     QStringList audioDevices,
00060                 numChannels,
00061                 cameraDevices,
00062                 displayResolutions;
00063
00064     QMap<QString,QStringList> sampleRates,cameraResolutions;
00065
00066     QComboBox *deviceComboBox,
00067                 *sampleRateComboBox,
00068                 *cameraDeviceComboBox,
00069                 *resolutionComboBox,
00070                 *bufferSizeComboBox,
00071                 *quantizationComboBox,
00072                 *frameRateComboBox,
00073                 *displayResolutionComboBox,
00074                 *displayFrameRateComboBox,
00075                 *frequencyRangeComboBox,
00076                 *roiComboBox,
00077                 *triggerComboBox,
00078                 *trackingAlgorithmComboBox;
00079
00080     QLabel *audioDeviceLabel,
00081            *sampleRateLabel,
00082            *cameraDeviceLabel,
00083            *cameraResolutionLabel,
00084            *bufferSizeLabel,
00085            *quantizationLabel,
00086            *numOutputChannelsLabel,
00087            *numOutputChannelsValue,
00088            *cameraFrameRateLabel,
00089            *screenResolutionLabel,
00090            *screenFrameRateLabel,
00091            *iavFrequencyRangeLabel,
00092            *iavRegionOfInterestLabel,
00093            *iavTriggerLabel,
00094            *iavTrackingAlgLabel,
00095            *accuracyLabel,
00096            *cpuLoadLabel,
00097            *errorLabel;
00098
00099     QSlider* skipFramesSlider;  // Horizontal slider
00100
00101     QHBoxLayout* numChannelsLayout;
00102 };
00103
00104 #endif
```

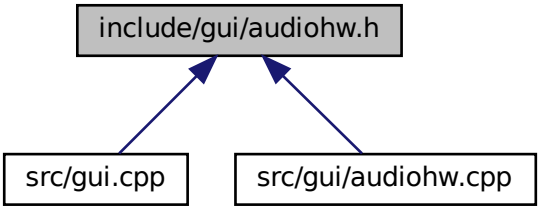
## 7.15 include/gui/audiohw.h File Reference

```
#include <utility>
#include <vector>
```

Include dependency graph for audiohw.h:




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




### Classes

- struct AudioHardware::Info

  *Structure representing audio hardware information.*

### Namespaces

- namespace AudioHardware

  *Audio hardware namespace provides functions to interact with audio hardware devices.*

### Functions

- const std::vector< unsigned int > AudioHardware::supportedRates ({8000, 11025, 16000, 22050, 32000, 44100, 48000, 88200, 96000, 176000, 192000, 352800, 384000})

  *List of supported sample rates.*
- bool AudioHardware::get_audio_device_info (int, int, std::pair< unsigned int, unsigned int > &, unsigned int &)

  *Retrieves information about available audio devices related to a specific audio card.*
- void AudioHardware::get_audio_hardware_info (std::vector< Info > &)

  *Retrieves information about available audio cards and their supported audio devices.*

### Variables

- const short int AudioHardware::MAX_POTENTIAL_AUDIO_DEVICES = 32

  *Custom defined - maximum number of potential audio devices that can be retrieved.*

- constexpr int AudioHardware::quantizationRatio { sizeof(float) * CHAR_BIT }

  *Bit size of the floating-point samples in bits.*

## 7.16 audiohw.h

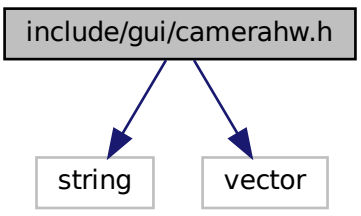
Go to the documentation of this file.

```
00001 #ifndef AUDIO_HW
00002 #define AUDIO_HW
00003
00004 #include <utility>
00005 #include <vector>
00006
00011 namespace AudioHardware{
00012
00016     const short int MAX_POTENTIAL_AUDIO_DEVICES = 32;
00017
00022     const std::vector<unsigned int> supportedRates({8000, 11025, 16000, 22050, 32000, 44100, 48000,
      88200, 96000, 176000, 192000, 352800, 384000});
00023
00027     constexpr int quantizationRatio { sizeof(float) * CHAR_BIT };
00028
00029     // using AHI=std::vector<std::pair< std::pair<std::string, std::string> , std::pair<unsigned int,
      unsigned int»>;
00030
00038     struct Info{
00039         std::pair<std::string, std::string> card_info;
00040         std::pair<unsigned int, unsigned int> sample_rate_range;
00041         unsigned int numberOfChannels;
00042     };
00043
00052     bool get_audio_device_info(int, int, std::pair<unsigned int, unsigned int>&, unsigned int&);
00053
00063     void get_audio_hardware_info(std::vector<Info>&);
00064
00065 }
00066
00067 #endif
```

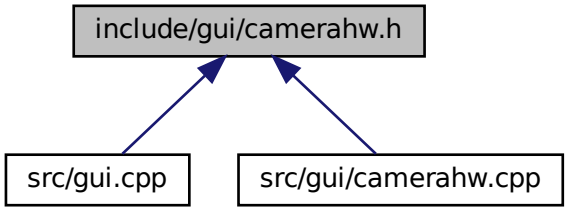
## 7.17 include/gui/camerahw.h File Reference

```
#include <string>
#include <vector>
```

Include dependency graph for camerahw.h:



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




## Classes

- struct CameraInfo

  *Represents information about a camera.*

## Functions

- std::vector< CameraInfo > getAvailableCameras ()

  *Retrieves information about available cameras on the system.*

## 7.17.1 Function Documentation

### 7.17.1.1 getAvailableCameras()

```
std::vector< CameraInfo > getAvailableCameras ( )
```

Retrieves information about available cameras on the system.
This function scans the system for connected cameras and gathers information about each camera, including its device path and supported resolutions.

**Returns**

A vector of CameraInfo structures, where each structure contains information about a single camera. If no cameras are found, an empty vector is returned.

Definition at line 15 of file camerahw.cpp.

```
00015                                           {
00016     std::vector<CameraInfo> cameras;
00017     std::unordered_set<std::string> uniquesResolutionValues;
00018
00019     for (int i = 0; i < 16; ++i) { // Check up to 16 potential camera devices
00020         std::string devicePath = "/dev/video" + std::to_string(i);
00021         int fd = open(devicePath.c_str(), O_RDWR | O_NONBLOCK, 0);
00022
00023         if (fd == -1) {
00024             if (errno == ENOENT || errno == EACCES) {
00025                 continue; // Device doesn't exist or no permission, try next
00026             } else {
00027                 perror("open");
00028                 continue; // Some other error, try next
00029             }
00030         }
00031
00032
00033         struct v4l2_capability cap;
00034         if (ioctl(fd, VIDIOC_QUERYCAP, &cap) == -1) {
00035             perror("VIDIOC_QUERYCAP");
00036             close(fd);
00037             continue;
```

```
00038         }
00039
00040         if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
00041              //Not a video capture device
00042             close(fd);
00043             continue;
00044         }
00045
00046         CameraInfo camera;
00047         camera.devicePath = devicePath;
00048
00049
00050
00051         struct v4l2_fmtdesc fmtdesc;
00052         memset(&fmtdesc, 0, sizeof(fmtdesc));
00053         fmtdesc.index = 0;
00054         fmtdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
00055
00056         while (ioctl(fd, VIDIOC_ENUM_FMT, &fmtdesc) == 0) {
00057             struct v4l2_frmsizeenum frmsizeenum;
00058             memset(&frmsizeenum, 0, sizeof(frmsizeenum));
00059             frmsizeenum.index = 0;
00060             frmsizeenum.pixel_format = fmtdesc.pixelformat;
00061
00062             while (ioctl(fd, VIDIOC_ENUM_FRAMESIZES, &frmsizeenum) == 0) {
00063
00064                 if (frmsizeenum.type == V4L2_FRMSIZE_TYPE_DISCRETE) {
00065                     std::string resVal = std::to_string(frmsizeenum.discrete.width) +"x"+
    std::to_string(frmsizeenum.discrete.height);
00066                     if(uniquesResolutionValues.find(resVal)== uniquesResolutionValues.end()){
00067                         camera.resolutions.push_back({frmsizeenum.discrete.width,
    frmsizeenum.discrete.height});
00068                         uniquesResolutionValues.insert(resVal);
00069                     }
00070                 } else if (frmsizeenum.type == V4L2_FRMSIZE_TYPE_STEPWISE) {
00071                      for (size_t w = frmsizeenum.stepwise.min_width; w <=
    frmsizeenum.stepwise.max_width; w += frmsizeenum.stepwise.step_width) {
00072                         for (size_t h = frmsizeenum.stepwise.min_height; h <=
    frmsizeenum.stepwise.max_height; h += frmsizeenum.stepwise.step_height) {
00073
00074                             std::string resVal = std::to_string(w) +"x"+ std::to_string(h);
00075                             if(uniquesResolutionValues.find(resVal)== uniquesResolutionValues.end()){
00076                                 camera.resolutions.push_back({w,h});
00077                                 uniquesResolutionValues.insert(resVal);
00078                             }
00079                         }
00080                     }
00081                 }
00082
00083                 frmsizeenum.index++;
00084             }
00085              fmtdesc.index++;
00086         }
00087
00088         std::sort(camera.resolutions.begin(),camera.resolutions.end(),compareResolutions);
00089         cameras.push_back(camera);
00090         close(fd);
00091     }
00092     return cameras;
00093 }
```

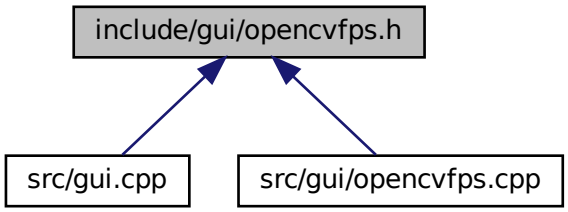
## 7.18 camerahw.h

Go to the documentation of this file.

```
00001 #ifndef CAMERA_HW
00002 #define CAMERA_HW
00003
00004 #include <string>
00005 #include <vector>
00006
00015 struct CameraInfo {
00016     std::string devicePath;
00017     std::vector<std::pair<int, int» resolutions;
00018 };
00019
00030 std::vector<CameraInfo> getAvailableCameras();
00031
00032 #endif
```

## 7.19 include/gui/opencvfps.h File Reference

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




### Functions

- double getCVfps_approx (const char *)

  *Calculates the camera's approximate frames per second (FPS) based on OpenCV's video capture.*

### 7.19.1 Function Documentation

#### 7.19.1.1 getCVfps_approx()

```
double getCVfps_approx (
            const char * cameraDevice )
```

Calculates the camera's approximate frames per second (FPS) based on OpenCV's video capture.

**Parameters**

| | |
|---|---|
| *const* | char* cameraDevice - The camera device id. |

**Returns**

double - The approximate frames per second (FPS) calculated from the video capture.

Definition at line 4 of file opencvfps.cpp.

```
00004                                           {
00005     cv::VideoCapture video(cameraDevice);
00006
00007     // Check camera
00008     // if (!video.isOpened()) {
00009     //     return -1;
00010     // }
00011
00012     return video.get(cv::CAP_PROP_FPS);
00013 }
```

## 7.20 opencvfps.h

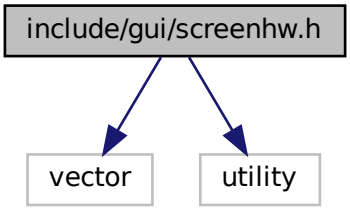
Go to the documentation of this file.

```
00001 #ifndef OPENCVFPS_H
00002 #define OPENCVFPS_H
00003
00010 double getCVfps_approx(const char*);
00011
00012 #endif
```

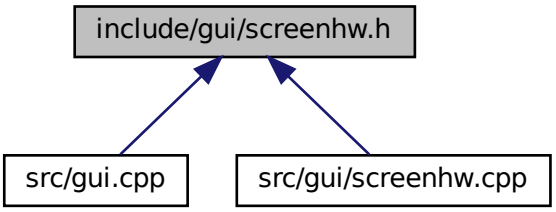
## 7.21 include/gui/screenhw.h File Reference

```
#include <vector>
#include <utility>
```

Include dependency graph for screenhw.h:




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




### Functions

- std::vector< std::pair< int, int > > get_screen_resolution ()

  *Retrieves the resolution of the primary (default) screen using platform-specific APIs.*

### 7.21.1 Function Documentation

#### 7.21.1.1 get_screen_resolution()

```
std::vector< std::pair< int, int > > get_screen_resolution ( )
```

Retrieves the resolution of the primary (default) screen using platform-specific APIs.

**Returns**

std::vector<std::pair<int,int>> - A list of pairs of integers representing the list of width and height supported by the screen.

Definition at line 10 of file screenhw.cpp.

```
00010                                                   {
00011
```

```
00012     std::vector<std::pair<int,int> screen_resolutions;
00013     std::unordered_set<std::string> uniquesValues;
00014 
00015     Display* display = XOpenDisplay(NULL);
00016     if (!display) {
00017         std::cerr « "Error:  Couldn't open display." « std::endl;
00018         return {};
00019     }
00020 
00021     int screen = DefaultScreen(display);
00022     Window root = RootWindow(display, screen);
00023 
00024     XRRScreenResources* resources = XRRGetScreenResourcesCurrent(display, root);
00025     if (!resources) {
00026         std::cerr « "Error:  Could not get screen resources." « std::endl;
00027         XCloseDisplay(display);
00028         return {};
00029     }
00030 
00031     // std::cout « "Available Screen Resolutions:" « std::endl;
00032 
00033     // Iterate through all modes and display valid resolutions
00034     for (int i = 0; i < resources->nmode; ++i) {
00035         XRRModeInfo* mode = &resources->modes[i];
00036         if (mode->width > 0 && mode->height > 0) { // Ensure valid dimensions
00037             // std::cout « mode->width « "x" « mode->height « std::endl;
00038             std::string resVal = std::to_string(mode->width) +"x"+ std::to_string(mode->height);
00039             if(uniquesValues.find(resVal)== uniquesValues.end()){
00040                 screen_resolutions.push_back({mode->width, mode->height});
00041                 uniquesValues.insert(resVal);
00042             }
00043         }
00044     }
00045 
00046     XRRFreeScreenResources(resources);
00047     XCloseDisplay(display);
00048 
00049     return screen_resolutions;
00050 }
```

## 7.22 screenhw.h

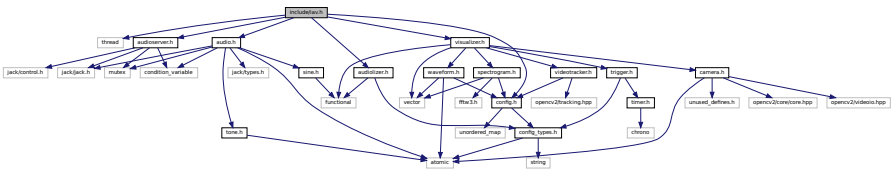
Go to the documentation of this file.

```
00001 #ifndef SCREENHW_H
00002 #define SCREENHW_H
00003 
00004 #include <vector>
00005 #include <utility>
00010 std::vector<std::pair<int,int» get_screen_resolution();
00011 
00012 #endif
00013 
```

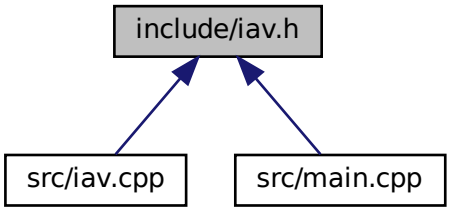
## 7.23 include/iav.h File Reference

```
#include <thread>
#include "config.h"
#include "audioserver.h"
#include "audio.h"
#include "audiolizer.h"
#include "visualizer.h"
```

Include dependency graph for iav.h:



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




## Classes

- class IAV

  *Class to manage the IAV multi-threaded processing pipeline.*

# 7.24 iav.h

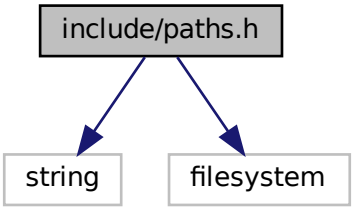
Go to the documentation of this file.

```
00001 #ifndef IAV_H
00002 #define IAV_H
00003
00004 #include <thread>
00005 #include "config.h"
00006 #include "audioserver.h"
00007 #include "audio.h"
00008 #include "audiolizer.h"
00009 #include "visualizer.h"
00010
00019 class IAV{
00020
00021     public:
00022
00026         IAV();
00027
00031         ~IAV();
00032
00037         void start();
00038
00042         IAV(const IAV&) = delete;
00043
00047         IAV(IAV&&) = delete;
00048
00052         IAV& operator=(const IAV&) = delete;
00053
00057         IAV&& operator=(IAV&&) = delete;
00058
00059     private:
00060
00061         Config& cfg = Config::getInstance();
00062         AudioServer audioServer;
00063         AudioStream audioStream;
00064         Audiolizer audiolizer;
00065         Visualizer visualizer;
00066
00067         std::thread audServerThread;
00068         std::thread audioThread;
00069         std::thread visualThread;
00070
00071         std::mutex mtxServer;
00072         std::condition_variable cvServer;
00073         bool serverStarted{false};
00074
00075 };
00076
00077
00078 #endif
```

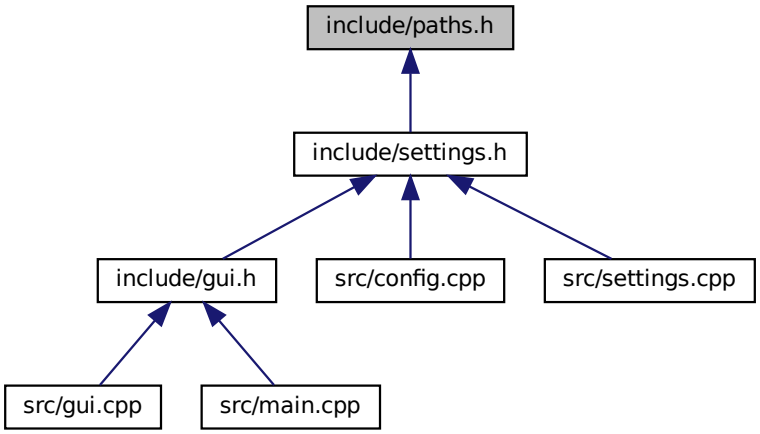
## 7.25 include/paths.h File Reference

```
#include <string>
#include <filesystem>
```

Include dependency graph for paths.h:




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




### Namespaces

- namespace Paths

  *Namespace containing the path to settings.db file.*
- namespace PathsTest

  *Namespace containing the path to the test.db file, used for testing purposes.*

### Functions

- auto getAbsPath (const std::string &relativePath) -> std::string

  *Function to get the absolute path of a given relative path.*

## Variables

- const std::string Paths::databasePath {getAbsPath("../data/settings.db")}
- const std::string PathsTest::databasePath {getAbsPath("../data/test.db")}

### 7.25.1 Function Documentation

#### 7.25.1.1 getAbsPath()

```
auto getAbsPath (
            const std::string & relativePath ) -> std::string   [inline]
```

Function to get the absolute path of a given relative path.

**Parameters**

| | |
|---|---|
| *relativePath* | - The path relative to the project's root directory. |

**Returns**

std::string - The absolute path of the given relative path.

**Note**

This function uses C++17's std::filesystem library.

**See also**

https://en.cppreference.com/w/cpp/filesystem/path

Definition at line 14 of file paths.h.

```
00014                                                                   {
00015     std::filesystem::path rootDir = std::filesystem::absolute(__FILE__).parent_path();
00016     std::filesystem::path targetPath = rootDir / relativePath;
00017     return targetPath.lexically_normal().string();
00018 }
```

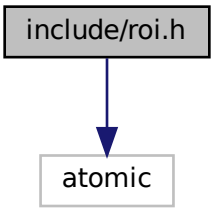
## 7.26 paths.h

Go to the documentation of this file.

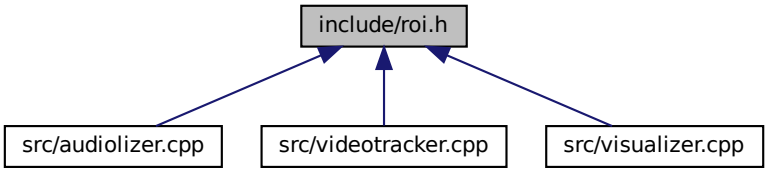
```
00001 #ifndef PATHS_H
00002 #define PATHS_H
00003
00004 #include <string>
00005 #include <filesystem>
00006
00014 inline auto getAbsPath(const std::string& relativePath) -> std::string {
00015     std::filesystem::path rootDir = std::filesystem::absolute(__FILE__).parent_path();
00016     std::filesystem::path targetPath = rootDir / relativePath;
00017     return targetPath.lexically_normal().string();
00018 }
00019
00024 namespace Paths{
00025     const std::string databasePath {getAbsPath("../data/settings.db")};
00026 }
00027
00035 namespace PathsTest{
00036     const std::string databasePath {getAbsPath("../data/test.db")};
00037 }
00038
00039 #endif
```

## 7.27 include/roi.h File Reference

`#include <atomic>`
Include dependency graph for roi.h:




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




### Classes

- struct RegionOfInterest

  *Struct to hold the region of interest (ROI) data.*

## 7.28 roi.h

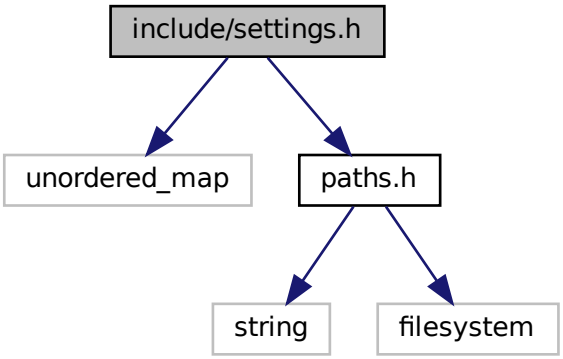
Go to the documentation of this file.

```
00001
00002 #ifndef ROI_H
00003 #define ROI_H
00004
00005 #include <atomic>
00006
00013 struct RegionOfInterest{
00015     std::atomic<int> centerX {0};
00017     std::atomic<int> centerY {0};
00019     std::atomic<int> volumeW {0};
00021     std::atomic<int> volumeH {0};
00022 };
00023
00024 #endif
```

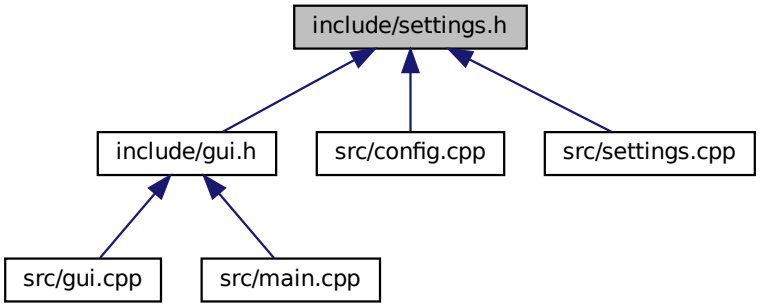
## 7.29 include/settings.h File Reference

```
#include <unordered_map>
#include "paths.h"
```

Include dependency graph for settings.h:




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




## Classes

- class SettingsDB

  *Class to manage settings using a SQLite database.*

# 7.30 settings.h

Go to the documentation of this file.

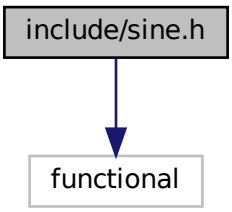
```
00001 #ifndef SETTINGS_H
00002 #define SETTINGS_H
00003
00004 #include <unordered_map>
00005 #include "paths.h"
00006 class sqlite3;
00007
00011 class SettingsDB {
00012 public:
00013
00019     explicit SettingsDB(const std::string& db_path=Paths::databasePath);
00020
00026     bool saveSettings(const std::unordered_map<std::string, std::string>& settings);
```

```
00027
00032     std::unordered_map<std::string, std::string> loadSettings();
00033
00037     ~SettingsDB();
00038
00039
00040     // complying with the rule of 5 to prevent unintented copying or moving.
00041
00045     SettingsDB(const SettingsDB&) = delete; // delete copy constructor
00046
00050     SettingsDB& operator=(const SettingsDB&) = delete; // delete copy assignment operator
00051
00055     SettingsDB(SettingsDB&&) = delete; // delete move constructor
00056
00060     SettingsDB& operator=(SettingsDB&&) = delete; //  delete move assignement operator
00061
00062 private:
00063     sqlite3* db;
00064     std::string dbPath;
00065 };
00066
00067
00068 #endif
```

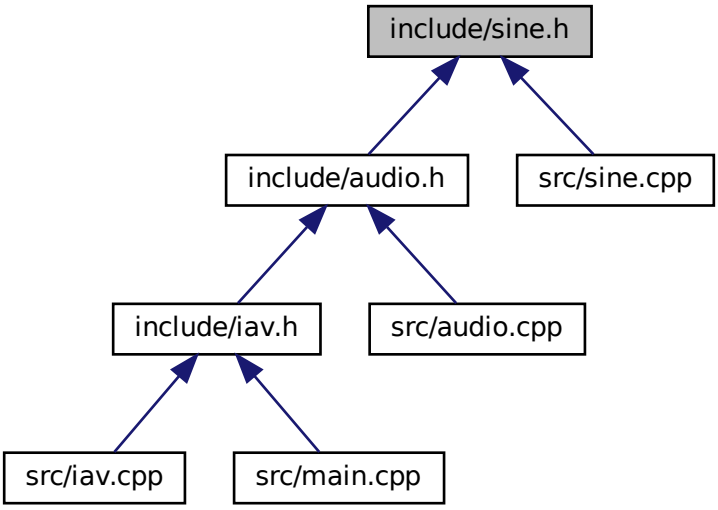
## 7.31 include/sine.h File Reference

`#include <functional>`

Include dependency graph for sine.h:



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




## Classes

- class Sine

  *Class responsible for generating sine wave signals for audio processing.*

# 7.32 sine.h

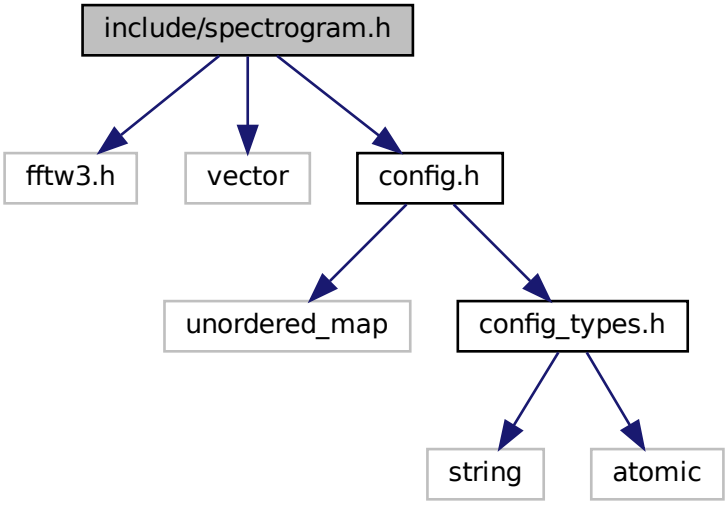
Go to the documentation of this file.

```
00001 #ifndef SINE_H
00002 #define SINE_H
00003
00004 #include <functional>
00005 class Config;
00006 struct AudioConfig;
00007 struct Tone;
00008
00012 class Sine{
00013
00014 public:
00015
00017     Sine();
00018
00024     void setVisualizerUpdater(std::function<void(float)>);
00025
00032     void setMonoSignal(Tone&, float*[2]);
00033
00039     void setStereoSignal(Tone&, float*[2]);
00040
00041 private:
00042     const AudioConfig& audiocfg;
00043     int prevfreq;
00044     float phase;
00045     float rads_per_sample;
00046
00047     std::function<void(float)> updateVisualizer;
00048 };
00049
00050 #endif
```

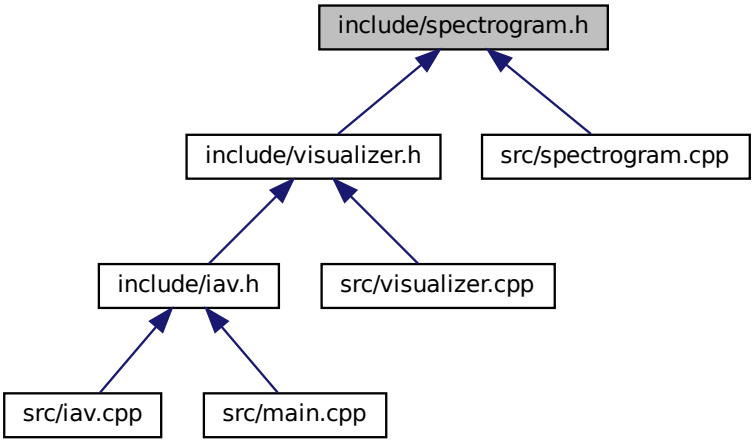
## 7.33 include/spectrogram.h File Reference

```
#include <fftw3.h>
#include <vector>
#include "config.h"
```

Include dependency graph for spectrogram.h:




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




### Classes

- class Spectrogram

  *Ring buffer class to generate a spectrogram of the audio signal using the Fast Fourier Transform (FFT).*

## Macros

- #define PI 3.14159

### 7.33.1 Macro Definition Documentation

#### 7.33.1.1 PI

```
#define PI 3.14159
```

Definition at line 9 of file spectrogram.h.

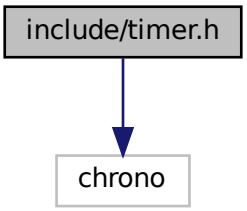
## 7.34 spectrogram.h

Go to the documentation of this file.

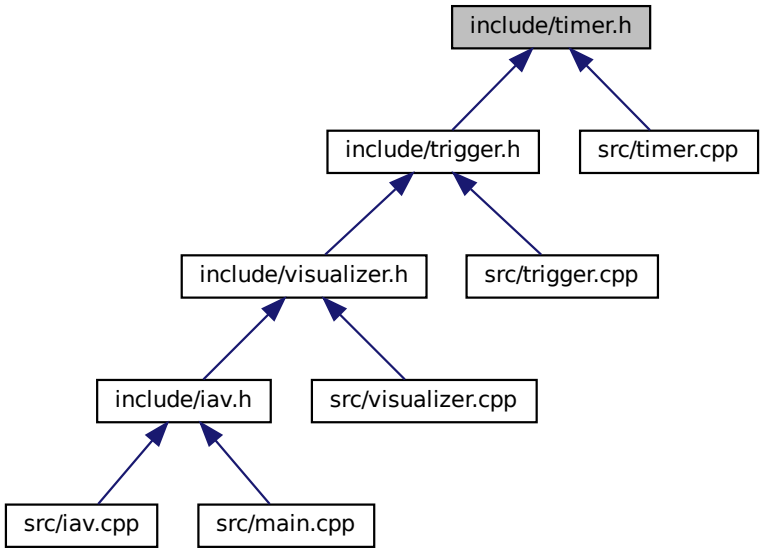
```
00001 #ifndef SPECTROGRAM_H
00002 #define SPECTROGRAM_H
00003
00004
00005 #include <fftw3.h>
00006 #include <vector>
00007 #include "config.h"
00008
00009 #define PI 3.14159
00010
00015 class Spectrogram{
00016 public:
00017
00021     Spectrogram();
00022
00026     ~Spectrogram();
00027
00031     Spectrogram (const Spectrogram&) = delete;
00032
00036     Spectrogram (Spectrogram&&) = delete;
00037
00041     Spectrogram& operator=(const Spectrogram&) = delete;
00042
00046     Spectrogram& operator=(Spectrogram&&) = delete;
00047
00053     int get_numAudioSamples();
00054
00059     int get_numFFTPoints();
00060
00066     bool write(const float&);
00067
00075     bool readBatch(std::vector<float>&, float&, float&);
00076
00077 private:
00078     Config &cfg = Config::getInstance();
00079
00080     int numAudioSamples, numFFTPoints;
00081     std::vector<float> ringBuffer,
00082         hamming_window;
00083
00084     std::atomic<size_t> readpos;
00085     std::atomic<size_t> writepos;
00086
00087     // fftw_complex *fft_in;
00088     std::vector<double> fft_in;
00089     fftw_complex *fft_out;
00090     fftw_plan plan;
00091
00092     float minMagnitude,maxMagnitude;
00093
00094     void calculateNFFT();
00095     void initialize_hamming(int);
00096
00097
00098 };
00099
00100 #endif
```

## 7.35 include/timer.h File Reference

`#include <chrono>`
Include dependency graph for timer.h:




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




### Classes

- class Timer

  *A class responsible for managing a timer.*

## 7.36 timer.h

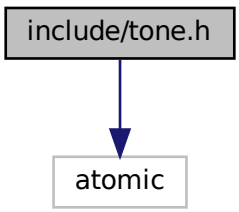
Go to the documentation of this file.

```
00001 #ifndef TIMER_H
00002 #define TIMER_H
00003
00004 #include <chrono>
```

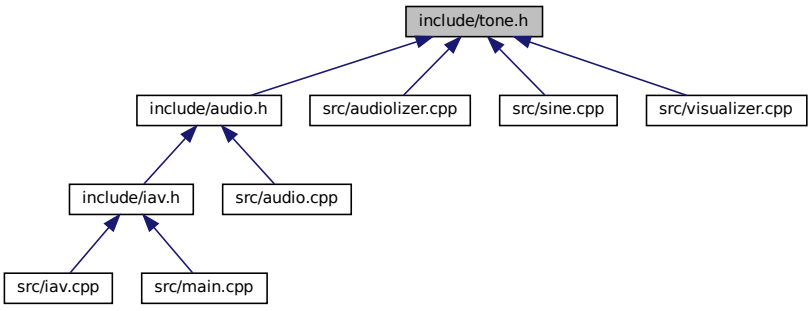
```
00005
00009 class Timer {
00010
00011     public:
00012
00016         Timer();
00017
00023         void setTimer(int);
00024
00029         void start();
00030
00035         bool update(int&);
00036
00041         int getRemainingTime() const;
00042
00047         bool isTimerFinished() const;
00048
00049     private:
00050
00051         int initialSeconds;                     // The initial time to start the countdown from (in
      seconds)
00052         int remainingTimeMilliseconds;          // Remaining time in milliseconds
00053         std::chrono::steady_clock::time_point lastUpdateTime;  // Last frame timestamp to calculate
      elapsed time
00054
00055 };
00056
00057
00058
00059 #endif
```

## 7.37 include/tone.h File Reference

`#include <atomic>`
Include dependency graph for tone.h:




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



### Classes

- struct Tone

  *A structure to represent a tone with its frequency and volume.*

## 7.38 tone.h

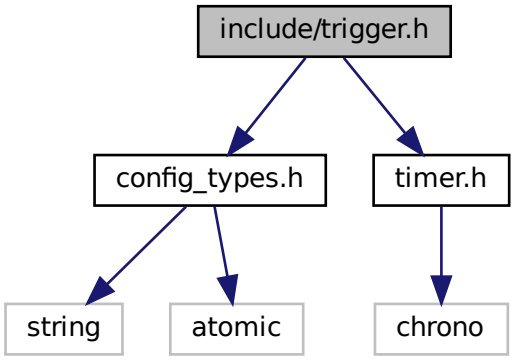
Go to the documentation of this file.

```
00001 #ifndef TONE_H
00002 #define TONE_H
00003
00004 #include <atomic>
00005
00012 struct Tone{
00014     std::atomic<int> frequency {0};
00016     std::atomic<float> volume {0.f};
00017 };
00018
00019
00020 #endif
```

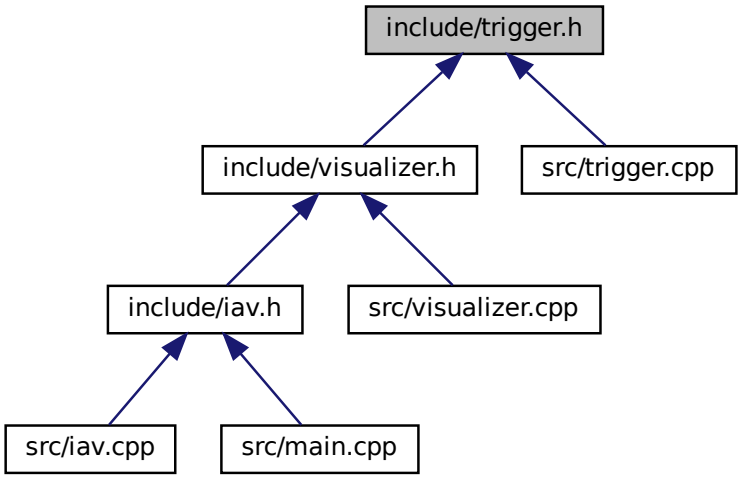
## 7.39 include/trigger.h File Reference

```
#include "config_types.h"
#include "timer.h"
```

Include dependency graph for trigger.h:



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




## Classes

- class Trigger

  *A class responsible for managing the trigger behavior.*

## Variables

- constexpr int experience_duration_sec = 10
- constexpr int photo_countdown_sec = 5

### 7.39.1 Variable Documentation

#### 7.39.1.1 experience_duration_sec

```
constexpr int experience_duration_sec = 10  [constexpr]
```

Definition at line 8 of file trigger.h.

#### 7.39.1.2 photo_countdown_sec

```
constexpr int photo_countdown_sec = 5  [constexpr]
```

Definition at line 9 of file trigger.h.

## 7.40 trigger.h

Go to the documentation of this file.

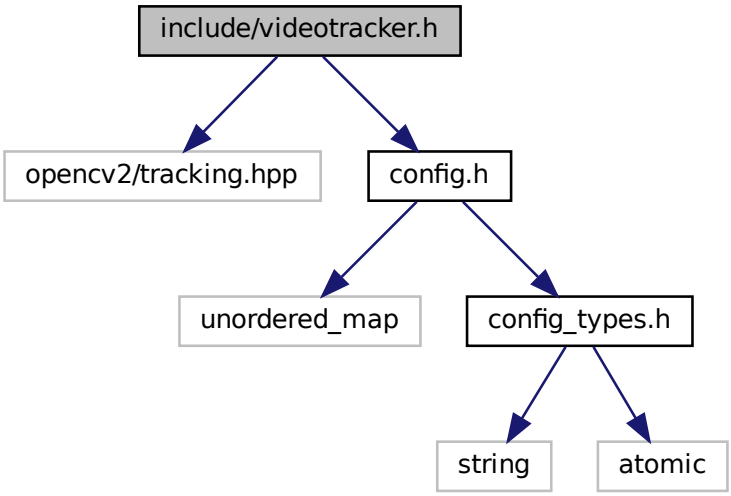
```
00001 #ifndef TRIGGER_H
00002 #define TRIGGER_H
00003
00004 #include "config_types.h"
00005 #include "timer.h"
00006 struct IAVConfig;
00007
```

```
00008 constexpr int experience_duration_sec = 10;
00009 constexpr int photo_countdown_sec = 5;
00010
00015 class Trigger {
00016
00017     public:
00018
00024         Trigger();
00025
00036         bool isTrackingEnabled(float&);
00037
00042         void reset();
00043
00049         Timer* getTimer();
00050     private:
00051         IAVConfig& iavcfg;
00052         Timer timer;
00053
00054         int timeDurationSec;
00055         bool mode; // tracking mode or not
00056
00057         void _modeToggle();
00058 };
00059
00060
00061
00062 #endif
```

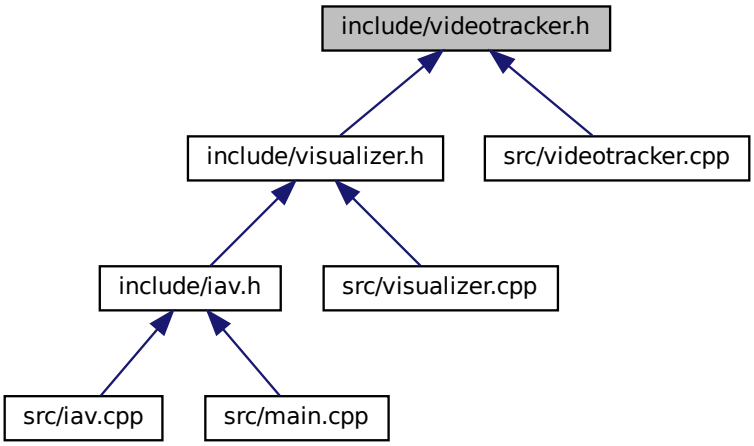
## 7.41 include/videotracker.h File Reference

```
#include <opencv2/tracking.hpp>
#include "config.h"
```

Include dependency graph for videotracker.h:



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




### Classes

- class VideoTracker

  *A class responsible for tracking objects in the camera feed.*

## 7.42 videotracker.h

Go to the documentation of this file.

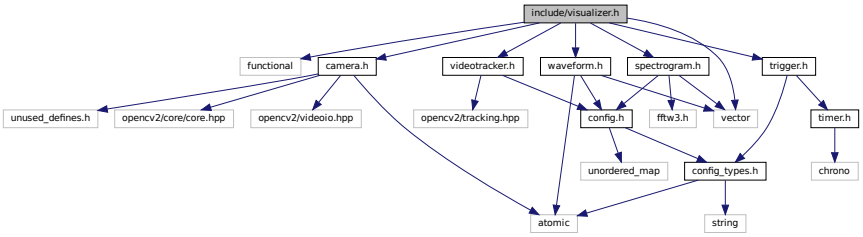
```
00001 #ifndef TRACKING_H
00002 #define TRACKING_H
00003
00004 #include <opencv2/tracking.hpp>
00005 #include "config.h"
00006 struct RegionOfInterest;
00007
00012 class VideoTracker{
00013 public:
00014
00016     VideoTracker();
00017
00023     void initializeTracker(const cv::Mat&);
00024
00031     bool trackObject(const cv::Mat&, RegionOfInterest&);
00032
00033 private:
00034
00035     Config &cfg = Config::getInstance();
00036
00037     cv::Ptr<cv::Tracker> tracker;
00038     cv::Rect centerBox;
00039     cv::Rect boundingBox;
00040
00041 };
00042
00043 #endif
```

## 7.43 include/visualizer.h File Reference

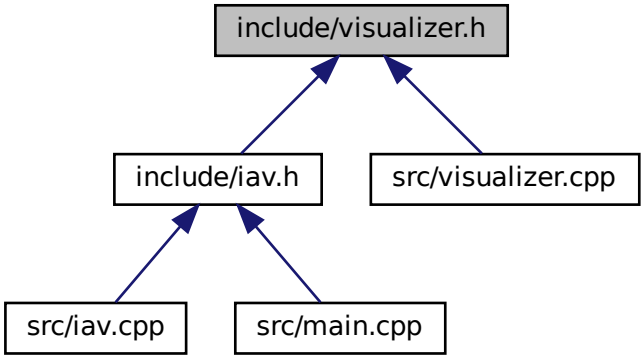
```
#include <functional>
#include <vector>
#include "camera.h"
#include "videotracker.h"
```

```
#include "trigger.h"
#include "waveform.h"
#include "spectrogram.h"
```

Include dependency graph for visualizer.h:




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




## Classes

- class Visualizer

  *This class is responsible for managing the camera feed, tracking objects, triggering, and broadcasting the visualized frame to the IAV pipeline.*

# 7.44 visualizer.h

Go to the documentation of this file.

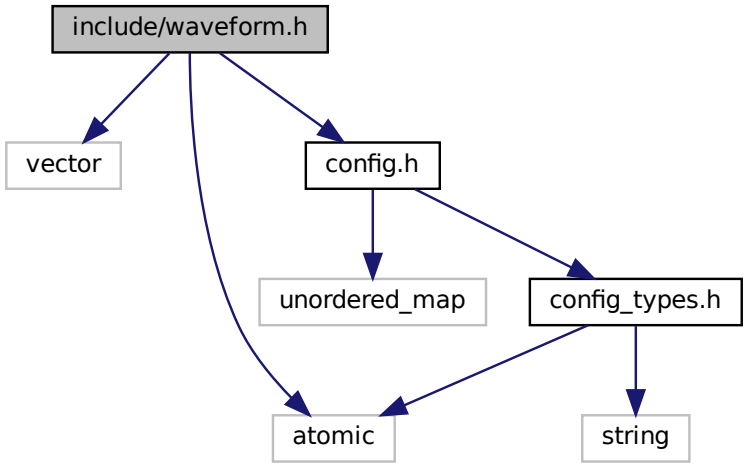
```
00001 #ifndef VISUALIZER_H
00002 #define VISUALIZER_H
00003
00004 #include <functional>
00005 #include <vector>
00006 #include "camera.h"
00007 #include "videotracker.h"
00008 #include "trigger.h"
00009 #include "waveform.h"
00010 #include "spectrogram.h"
00011 class RegionOfInterest;
00012 class Tone;
00013
00017 class Visualizer{
00018 public:
```

```
00019
00021     Visualizer();
00022
00024     ~Visualizer();
00025
00029     Visualizer(const Visualizer&) = delete;
00030
00034     Visualizer(Visualizer&&) = delete;
00035
00039     Visualizer& operator=(const Visualizer&) = delete;
00040
00044     Visualizer& operator=(Visualizer&&) = delete;
00045
00051     void broadcast();
00052
00062     void setAudiolizerUpdater(std::function<void(const bool, const bool, const RegionOfInterest&,
      Tone&)>);
00063
00069     void updateAudioSignal(float);
00070
00071 private:
00072
00073     const Config &cfg = Config::getInstance();
00074
00075     Camera camera;
00076     VideoTracker videoTracker;
00077     Trigger trigger;
00078     Waveform waveform;
00079     Spectrogram spectrogram;
00080     std::vector<float> specMagnitude;
00081
00082     cv::Mat visualFrame,cameraFrame;
00083     cv::Mat camBinaryMask;
00084
00085     float transpose_ratio_x, transpose_ratio_y;
00086     int R,G,B;
00087     int LR,TB;
00088     int numPointsPerimeter;
00089     int leftMidFreq, rightMidFreq;
00090
00091     void _setToCamera(float);
00092     void _show_timer(float);
00093
00094     bool trackingToggle;
00095     void updateTrackingMode(bool);
00096     std::function<void(const bool, const bool, const RegionOfInterest&, Tone&)> updateAudioLizer ;
00097
00098     void _set_BG_manually(Tone&);
00099     void _set_FG_manually(const RegionOfInterest&);
00100     void _create_camMask();
00101     bool _showFrame();
00102     void _set_freq_midBoundaries();
00103
00104     void draWaveform();
00105     void drawSpectrogram();
00106     void drawSmallcircle(const RegionOfInterest &);
00107 };
00108
00109
00110 #endif
```

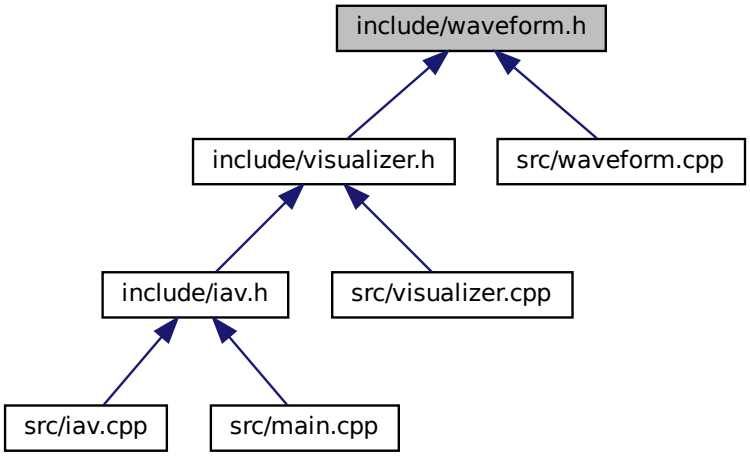
## 7.45 include/waveform.h File Reference

```
#include <vector>
#include <atomic>
#include "config.h"
```

Include dependency graph for waveform.h:




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




## Classes

- class Waveform

  *A circular buffer for storing audio samples.*

# 7.46 waveform.h

Go to the documentation of this file.

```
00001 #ifndef RAW_H
```

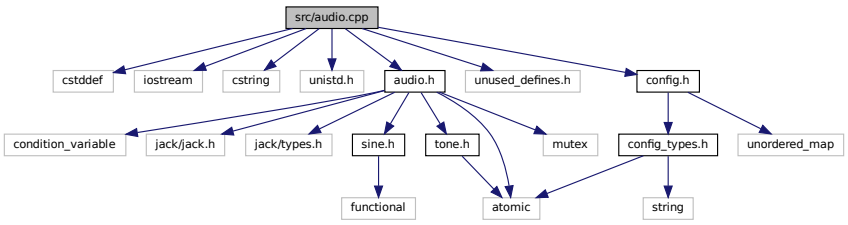
```
00002 #define RAW_H
00003
00004 #include <vector>
00005 #include <atomic>
00006 #include "config.h"
00007
00012 class Waveform{
00013 public:
00014
00016     Waveform();
00017
00023     bool write(const float&);
00024
00030     bool read(float&);
00031
00036     bool isEmpty() const;
00037
00038     /*
00039 * @brief Checks if the buffer is full.
00040 * @return bool - true if the buffer is full, false otherwise.
00041 */
00042     bool isFull() const;
00043
00048     size_t size() const;
00049
00054     size_t availableForReading() const;
00055
00061     void getMinMax(float[2]);
00062
00063 private:
00064
00065     Config &cfg = Config::getInstance();
00066     std::vector<float> waveTable; // The actual buffer;
00067     std::atomic<size_t> readpos;
00068     std::atomic<size_t> writepos;
00069     float min,max;
00070     size_t capacity;
00071 };
00072
00073 #endif
```

## 7.47 README.md File Reference

## 7.48 src/audio.cpp File Reference

```
#include <cstddef>
#include <iostream>
#include <cstring>
#include <unistd.h>
#include "audio.h"
#include "unused_defines.h"
#include "config.h"
```

Include dependency graph for audio.cpp:




### Variables

- const char * server_name = nullptr
- const char clientName [17] = "IAV-audio-client"

### 7.48.1 Variable Documentation

#### 7.48.1.1 clientName

```
const char clientName[17] = "IAV-audio-client"
```

Definition at line 11 of file audio.cpp.

#### 7.48.1.2 server_name

```
const char* server_name = nullptr
```

Definition at line 10 of file audio.cpp.

## 7.49 audio.cpp

Go to the documentation of this file.

```
00001 #include <cstddef>
00002 #include <iostream>
00003 #include <cstring>
00004 #include <unistd.h>
00005 #include "audio.h"
00006 #include "unused_defines.h"
00007 #include "config.h"
00008
00009
00010 const char *server_name = nullptr;
00011 const char clientName[17] = "IAV-audio-client";
00012
00013 int AudioStream::streamAudio ( jack_nframes_t UNUSED(nframes), void *arg){ //, float *in,void
      (*threading)(float *sig)
00014
00015     return static_cast<AudioStream*>(arg)->streamBuffer();
00016 }
00017
00018 AudioStream::AudioStream():audiocfg (Config::getInstance().audconf){
00019
00020     client_name=clientName;
00021     // nullify all
00022     client = nullptr;
00023     todevice = nullptr;
00024
00025     if (audiocfg.numChannels.load() == 1){
00026         output_ports[1] = nullptr;
00027         outputBuffers[1]=nullptr;
00028         make_sound = &Sine::setMonoSignal;  // Point to setMonoSignal for processing 1 single mono
      buffer
00029     }
00030     else if (audiocfg.numChannels.load() == 2) {
00031         make_sound = &Sine::setStereoSignal;  // Point to setStereoSignal for processing 2 stereo
      buffers
00032     }
00033 }
00034
00035 void AudioStream::setVisualizerUpdater(std::function<void(float)> updater){
00036     sine.setVisualizerUpdater(std::move(updater));
00037 }
00038
00039 AudioStream::~AudioStream(){
00040     closeStream();
00041     std::cout«"Audio stream object destructed"«std::endl;
00042 }
00043
00044 void AudioStream::clientConnect(std::mutex& mtx, std::condition_variable& cv, bool& serverStarted){
00045
00046     std::cout « "Waiting for jack server to start\n";
00047     std::unique_lock<std::mutex> lock(mtx);
00048     cv.wait(lock, [&] { return serverStarted; });
00049
00050     jack_options_t options =
      JackNoStartServer;//(JackSessionID|JackServerName|JackNoStartServer|JackUseExactName|JackNullOption)
00051     jack_status_t status;
00052
00053     /* open a client connection to the JACK server */
00054     client = jack_client_open (client_name, options, &status,nullptr);
00055     if (status & JackNameNotUnique) {    //client name not unique, set a client name;
00056         client_name = jack_get_client_name(client);
00057         std::cerr«"\t»unique name "«client_name«" assigned to the client obj."«std::endl;
```

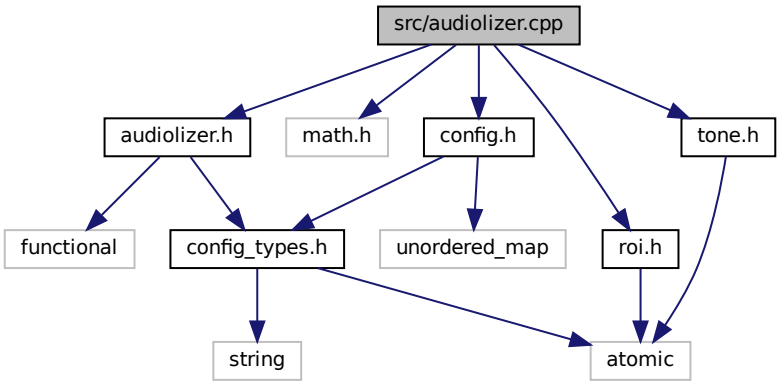
```
00058     }
00059
00060     if (client == NULL) {
00061         std::cerr«"\t»jack_client_open() failed, status = "«status«std::endl;
00062         if (status & JackServerFailed) {
00063             std::cerr«"\t»Unable to connect to JACK server"«std::endl;
00064         }
00065         exit (1);
00066     }
00067     if (status & JackServerStarted) {
00068         std::cout«"\t»JACK server started"«std::endl;
00069     }
00070
00071     //callback
00072     if (jack_set_process_callback (client,streamAudio,this)){ //arg
00073             std::cerr«"\t»Callback operation failed"«std::endl;
00074     }
00075
00076     //prevent failure
00077     jack_on_shutdown(client,&jack_shutdown,0);
00078
00079     //register physical ports
00080     for (size_t ch=0; ch<audiocfg.numChannels.load();++ch){
00081         std::string portName = (ch%2) ?  ("PortRight"+std::to_string(ch/2)) :
      ("PortLeft"+std::to_string(ch/2));
00082         // std::cout«"portName = "«portName«std::endl;
00083         output_ports[ch]=jack_port_register (client,portName.c_str(),JACK_DEFAULT_AUDIO_TYPE,
      JackPortIsOutput, 0);
00084         // output_port_right=jack_port_register (client,"rightPort",JACK_DEFAULT_AUDIO_TYPE,
      JackPortIsOutput, 0);
00085         if (output_ports[ch] == NULL){
00086             std::cerr«"\t»Unable to register output port for
      {"«jack_port_name(output_ports[ch])«"}"«std::endl;
00087             exit (1);}
00088     }
00089
00090     //activate client
00091     if (jack_activate (client)) {
00092         std::cerr«"\t»cannot activate client {"«client_name«"}"«std::endl;
00093         exit (1);
00094     }
00095
00096     // Getting acces to destination ports
00097     todevice = jack_get_ports (client, NULL, NULL, JackPortIsPhysical|JackPortIsInput);
00098     if (todevice == NULL) {
00099         std::cerr«"\t»no physical playback devices"«std::endl;
00100         exit (1);
00101     }
00102
00103     for (size_t ch=0; ch <audiocfg.numChannels.load();++ch){
00104         if (output_ports[ch]!=NULL){
00105             if (jack_connect (client, jack_port_name(output_ports[ch]), todevice[ch])){//returns full
      name
00106                 std::cerr«"\t»cannot connect left plysical output port {"«todevice[ch]«"} with input
      port {"«jack_port_name(output_ports[ch])«"}"«std::endl;
00107             }
00108         }
00109     }
00110
00111     free (todevice);
00112 }
00113
00114
00115 void AudioStream::closeStream(){
00116
00117     for (size_t i=0; i<audiocfg.numChannels.load();++i){
00118         if (jack_port_connected(output_ports[i])){
00119             if(jack_port_disconnect(client,output_ports[i])){
00120                 std::cerr«"Couldnt disconnect the "«jack_port_name(output_ports[i])«" output port from
      the main stream"«std::endl;
00121             }
00122         }
00123
00124     }
00125
00126     std::cout«"Closing stream - turning off audio client.."«std::endl;
00127     jack_client_close (client);
00128 }
00129
00130 int AudioStream::streamBuffer(){
00131
00132     for (size_t ch = 0 ; ch < audiocfg.numChannels.load(); ++ch){
00133         outputBuffers[ch] = static_cast<float *>(jack_port_get_buffer (output_ports[ch],
      audiocfg.bufferSize.load() ));
00134     }
00135
00136     (sine.*make_sound)(tone,outputBuffers);
```

```
00137
00138
00139     return 0;
00140 }
00141
00142 void AudioStream::jack_shutdown (void *UNUSED(arg))
00143 {
00144     exit (1);
00145 }
00146
00152 void AudioStream::update(int frequency, float volume){
00153     tone.frequency.store(frequency);
00154     tone.volume.store(volume);
00155 }
```

## 7.50 src/audiolizer.cpp File Reference

```
#include "audiolizer.h"
#include <math.h>
#include "config.h"
#include "roi.h"
#include "tone.h"
```

Include dependency graph for audiolizer.cpp:




## 7.51 audiolizer.cpp

Go to the documentation of this file.

```
00001 #include "audiolizer.h"
00002 #include <math.h>
00003 #include "config.h"
00004 #include "roi.h"
00005 #include "tone.h"
00006
00007
      Audiolizer::Audiolizer():cameracfg(Config::getInstance().camconf),iavcfg(Config::getInstance().iavconf){
00008
00009     // @TEMPORARY DISABLED
00010     // init_log_freq_scale(); // currently not used.  Use _int2log_freq (currently not used either) is
      affected by this method..
00011
00012     frequencyRange = iavcfg.maxFrequency - iavcfg.minFrequency;
00013     prev_freq=0;
00014     volume = 0.f;
00015 }
00016
00017 void Audiolizer::setAudioUpdater(std::function<void(int, float)> func){
00018     updateAudio = std::move(func);
00019 }
00020
```

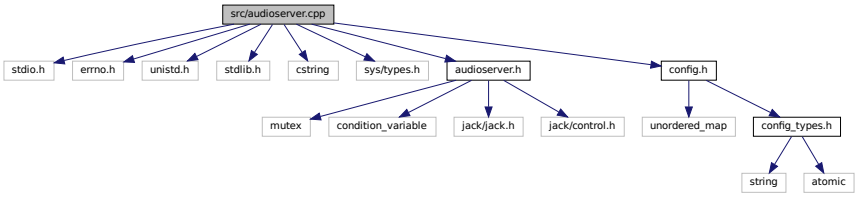
```
00021 bool Audiolizer::turn_Image_into_Sound(const bool tracking_updated, const bool pattern_locked , const
      RegionOfInterest &roi, Tone &tone){
00022
00023 /***
00024 * returns by reference the frequency that will be streamed on the next audio buffer
00025 */
00026
00027     int frequency = tone.frequency.load();
00028     int prevFreq = prev_freq;
00029
00030     if (pattern_locked){
00031         if (tracking_updated)          // if tracking updated --> new x,y --> new freq
00032             translate(roi, frequency);
00033         else{                          // else --> previous frequency
00034             frequency=prev_freq;
00035         }
00036     }else{                            // gradualy fade frequency to zero --> if frequency > 0 , slowly
      decline
00037         if (frequency>1){
00038             gradualy_fade(frequency); // gradualy fade frequency to zero --> if frequency > 0 , slowly
      decline
00039         }else{
00040             frequency=0;
00041             volume = 0.f;
00042         }
00043     }
00044
00045     // update audioStream with the newFrequency
00046     bool frequencyChanged = frequency != prevFreq;
00047     if (frequencyChanged){
00048         updateAudio(frequency , volume);
00049     }
00050     tone.frequency.store(frequency);
00051     tone.volume.store(volume);
00052
00053     return frequencyChanged;
00054
00055 }
00056
00057 bool Audiolizer::translate(const RegionOfInterest &roi, int& freq){
00058
00059     // translate the x coordinate.
00060     float roiCenterX = static_cast<float>(roi.centerX.load());
00061     // normalize x, y position
00062     float spatial_percent = roiCenterX / static_cast<float>(cameracfg.camResW.load());
00063     //apply translation from x,y to Hz
00064     freq = iavcfg.minFrequency + static_cast<int>(spatial_percent* static_cast<float>( frequencyRange
      ));
00065     // @TEMPORARY DISABLED
00066     // int2log_freq(freq); // define here the logarthmic tranformation of the input freq
00067
00068     // translate the y coordinate
00069     float roiCenterY = static_cast<float>(roi.centerY.load());
00070     // volume ranges from 0.1 up to 0.7 ==> percentage=0.1+(sample*(0.70.1)/maxVal)
00071     // volume = 0.1f + (( roiCenterY * 0.6f) / static_cast<float>(cameracfg.camResH.load()));
00072     // volume ranges from 1.0 down to 0.1 ==> percentage=1.0 - (sample*(1.00.1)/maxVal)
00073     volume= 1.0f - (( roiCenterY * 0.9f) / static_cast<float>(cameracfg.camResH.load()));
00074
00075     if (freq!=prev_freq){ // if previous frequency has the same value as before it returns the
      previous frequency
00076         prev_freq = freq;
00077         return true;
00078     }else
00079         return false;
00080 }
00081
00082 void Audiolizer::gradualy_fade(int& freq){
00083     if (freq>( frequencyRange )/2) freq -= static_cast<int>(2*log(freq));
00084     else if (freq<( frequencyRange )/2) freq -= static_cast<int>(log(freq));
00085     else if (freq<0) freq=0;
00086 }
00087
00088
00089 /* @TEMPORARY DISABLED
00090 void Audiolizer::init_log_freq_scale(){
00091
00092 double minfreq = static_cast<double>(iavcfg.minFrequency);
00093 double maxfreq = static_cast<double>(iavcfg.maxFrequency);
00094 double maxW = static_cast<double>(cameracfg.camResW.load());
00095
00096 // b = log (y2/y1) / (x2-x1) ---> where x1 (minW ==0), x2 (maxW), y1 (minFreq) and y2 (maxFreq)
00097 b = log (maxfreq / minfreq) / (double)(maxW - 0);
00098 // a = y2 / exp bx2
00099 a = maxfreq / (exp(b*(double)maxW));
00100
00101 // given x, find log freq by solving :  y = a exp bx
00102 // ...    (    definition in this->_int2log_freq()     )
```

```
00103 }
00104
00105 void Audiolizer::int2log_freq(int &freq){
00106 // given x, find log freq by solving :  y = a exp bx
00107 freq = a * exp(b * static_cast<double>(freq));
00108 }
00109 */
```

## 7.52 src/audioserver.cpp File Reference

```
#include <stdio.h>
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <cstring>
#include <sys/types.h>
#include "audioserver.h"
#include "config.h"
```

Include dependency graph for audioserver.cpp:




## 7.53 audioserver.cpp

Go to the documentation of this file.

```
00001 #include <stdio.h>
00002 #include <errno.h>
00003 #include <unistd.h>
00004 #include <stdlib.h>
00005 #include <cstring>
00006 #include <sys/types.h>
00007 #include "audioserver.h"
00008 #include "config.h"
00009
00010 AudioServer::AudioServer(const char* driverName):driver_name(driverName),audiocfg
      (Config::getInstance().audconf){
00011     server = jackctl_server_create2(NULL, NULL, NULL);
00012     parameters = jackctl_server_get_parameters(server);
00013     sigmask = jackctl_setup_signals(0);
00014     drivers = jackctl_server_get_drivers_list(server);
00015
00016 }
00017
00018 AudioServer::~AudioServer(){
00019     printf("Stopping server\n");
00020     stop_server();
00021 }
00022
00023 void AudioServer::setup_server(){
00024
00025     change_server_parameters();
00026 #ifdef SERVER_VERBOSE
00027     print_driver_info();
00028 #endif
00029     change_ALSAdriver_parameters();
00030 }
00031 void AudioServer::stop_server(){
00032     printf("\n\nShutting down server\n\n");
00033     jackctl_server_stop(server);
00034     jackctl_server_close(server);
00035     jackctl_server_destroy(server);
00036 }
00037 void AudioServer::start_server(std::mutex& mtx, std::condition_variable& cv, bool& serverStarted){
```

```
00038     jackctl_server_open(server, jackctl_server_get_driver());
00039     jackctl_server_start(server);
00040 
00041     // Signal that server has been started
00042     {
00043         std::lock_guard<std::mutex> lock(mtx);
00044         serverStarted = true;
00045     }
00046     cv.notify_one();
00047 
00048     jackctl_wait_signals(sigmask);
00049 }
00050 void AudioServer::change_server_parameters(){
00051     // change server param --> make verbose
00052     jackctl_parameter_t* param;
00053     union jackctl_parameter_value value;
00054     param = jackctl_get_parameter(parameters, "verbose");
00055     if (param != NULL) {
00056         value.b = false;//true;
00057         jackctl_parameter_set_value(param, &value);
00058     }
00059     // change server param --> make real-time
00060     param = jackctl_get_parameter(parameters, "realtime");
00061     if (param != NULL) {
00062         value.b = true;
00063         jackctl_parameter_set_value(param, &value);
00064         printf("Success on changing real time");
00065     }
00066     // change server param --> change real-time priority
00067     param = jackctl_get_parameter(parameters, "realtime-priority");
00068     if (param != NULL) {
00069         value.b = 80;
00070         jackctl_parameter_set_value(param, &value);
00071         printf("Success on changing real-time priority");
00072     }
00073 }
00074 
00075 jackctl_driver_t* AudioServer::jackctl_server_get_driver()
00076 {
00077     const JSList * node_ptr = drivers;
00078     while (node_ptr) {
00079         if (strcmp(jackctl_driver_get_name(static_cast<jackctl_driver_t *>(node_ptr->data)),
    driver_name) == 0) {
00080             return (jackctl_driver_t *)node_ptr->data;
00081         }
00082         node_ptr = jack_slist_next(node_ptr);
00083     }
00084     return NULL;
00085 }
00086 jackctl_parameter_t* AudioServer::jackctl_get_parameter(const JSList * parameters_list,const char *
    parameter_name){
00087     while (parameters_list)
00088     {
00089         if (strcmp(jackctl_parameter_get_name(static_cast<jackctl_parameter_t
    *>(parameters_list->data)), parameter_name) == 0)
00090         {
00091             return (jackctl_parameter_t *)parameters_list->data;
00092         }
00093         parameters_list = jack_slist_next(parameters_list);
00094     }
00095     return NULL;
00096 }
00097 
00098 #ifdef SERVER_VERBOSE
00099 
00100 void AudioServer::print_parameters(const JSList * node_ptr)
00101 {
00102     while (node_ptr != NULL) {
00103         jackctl_parameter_t * parameter = static_cast<jackctl_parameter_t*>(node_ptr->data);
00104         printf("\nparameter name = %s\n", jackctl_parameter_get_name(parameter));
00105         printf("parameter id = %c\n", jackctl_parameter_get_id(parameter));
00106         printf("parameter short decs = %s\n", jackctl_parameter_get_short_description(parameter));
00107         printf("parameter long decs = %s\n", jackctl_parameter_get_long_description(parameter));
00108         print_value(jackctl_parameter_get_default_value(parameter),
    jackctl_parameter_get_type(parameter));
00109         node_ptr = jack_slist_next(node_ptr);
00110     }
00111 }
00112 
00113 void AudioServer::print_value(union jackctl_parameter_value value, jackctl_param_type_t type){
00114     switch (type) {
00115 
00116         case JackParamInt:
00117             printf("parameter value = %d\n", value.i);
00118             break;
00119 
00120          case JackParamUInt:
```

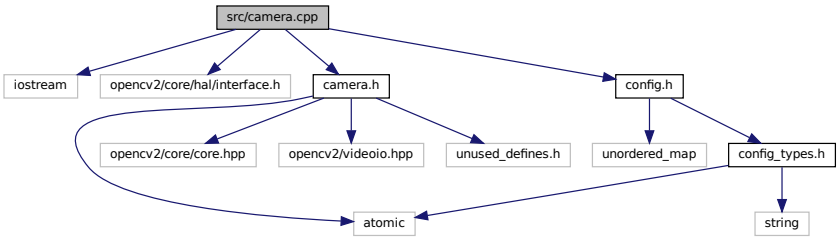
```
00121             printf("parameter value = %u\n", value.ui);
00122             break;
00123
00124          case JackParamChar:
00125             printf("parameter value = %c\n", value.c);
00126             break;
00127
00128          case JackParamString:
00129             printf("parameter value = %s\n", value.str);
00130             break;
00131
00132          case JackParamBool:
00133             printf("parameter value = %d\n", value.b);
00134             break;
00135      }
00136 }
00137
00138 void AudioServer::print_driver_info(){
00139     const JSList * node_ptr = drivers;
00140     while (node_ptr != NULL) {
00141         jackctl_driver_t *driver = static_cast<jackctl_driver_t *>(node_ptr->data);
00142         if (!strcmp(jackctl_driver_get_name(driver),driver_name)){
00143             printf("\n--------------------------\n");
00144             printf("driver = %s\n", jackctl_driver_get_name(driver));
00145             printf("-------------------------- \n");
00146             print_parameters(jackctl_driver_get_parameters(driver));
00147         }
00148         node_ptr = jack_slist_next(node_ptr);
00149     }
00150 }
00151 #endif
00152
00153 void AudioServer::change_ALSAdriver_parameters(){
00154     const JSList * node_ptr = drivers;
00155     while (node_ptr != NULL) {
00156
00157         jackctl_driver_t *driver = static_cast<jackctl_driver_t *>(node_ptr->data);
00158         if (!strcmp(jackctl_driver_get_name(driver),driver_name)){
00159                     const JSList * param_ptr = jackctl_driver_get_parameters(driver);
00160                     while (param_ptr != NULL) {
00161
00162                         jackctl_parameter_t * parameter = static_cast<jackctl_parameter_t
      *>(param_ptr->data);
00163                         const char* param_name = jackctl_parameter_get_name(parameter);
00164                         // Configure sample rate
00165                         if (!strcmp(param_name,"rate")){
00166                             int sr = audiocfg.sampleRate.load();
00167                             if (jackctl_parameter_set_value (parameter, (const union
      jackctl_parameter_value*)&sr)){
00168                                 printf("Audioserver::change_ALSAdriver_parameters :  sample rate
      changed succesfully to %d\n",sr);
00169                             }
00170                             // else{
00171                             //     jackctl_parameter_value jpv =
      jackctl_parameter_get_value(parameter);
00172                             //     cfg.audconf.sampleRate.store(static_cast<int>(jpv.ui));
00173                             //     printf("Audioserver::change_ALSAdriver_parameters :  Reconfiguring
      sample rate to %d\n",cfg.audconf.sampleRate.load());
00174                             // }
00175                         }
00176                         // Configure device name
00177                         else if (!strcmp(param_name,"device")){
00178                             std::string device_name_str = "hw:"+audiocfg.audioDevice;
00179                             const char* device_name = device_name_str.c_str();
00180                             if (jackctl_parameter_set_value (parameter, (const union
      jackctl_parameter_value*)device_name )){
00181                                 printf("Audioserver::change_ALSAdriver_parameters :  device name has
      changed to:  %s\n",device_name);
00182                             }
00183                             // else{
00184                             //     jackctl_parameter_value defaultDevice =
      jackctl_parameter_get_default_value(parameter);
00185                             //     cfg.audconf.audioDevice = defaultDevice.str;
00186                             //     printf("Audioserver::change_ALSAdriver_parameters :  configuring
      default device to :    %s\n",cfg.audconf.audioDevice.c_str());
00187                             // }
00188                         }
00189                         // Configure buffer size
00190                         else if (!strcmp(param_name,"period")){
00191                             int buffer_size = audiocfg.bufferSize.load();
00192                             if (jackctl_parameter_set_value (parameter, (const union
      jackctl_parameter_value*)&buffer_size)){
00193                                 printf("Audioserver::change_ALSAdriver_parameters :  buffer size has
      changed to:  %d \n",buffer_size);
00194                             }
00195                             // else{
00196                             //     jackctl_parameter_value jpv =
```

```
    jackctl_parameter_get_value(parameter);
00197                           //     cfg.audconf.bufferSize.store(static_cast<int>(jpv.ui));
00198                           //     printf("Audioserver::change_ALSAdriver_parameters :  buffer size
    has NOT changed.  Current buffer size value :  %d \n",cfg.audconf.bufferSize.load());
00199                           // }
00200                       }
00201                       param_ptr = jack_slist_next(param_ptr);
00202                   }
00203           }
00204           node_ptr = jack_slist_next(node_ptr);
00205       }
00206 }
```

## 7.54 src/camera.cpp File Reference

```
#include <iostream>
#include <opencv2/core/hal/interface.h>
#include "camera.h"
#include "config.h"
```

Include dependency graph for camera.cpp:




## 7.55 camera.cpp

Go to the documentation of this file.

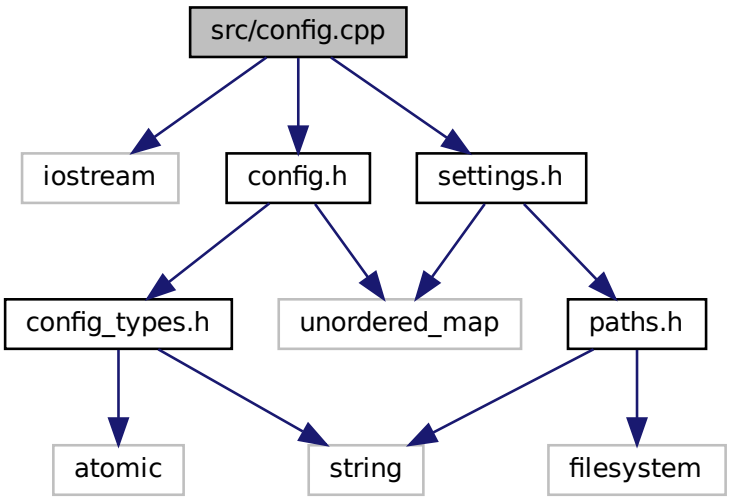
```
00001 #include <iostream>
00002 #include <opencv2/core/hal/interface.h>
00003 #include "camera.h"
00004 #include "config.h"
00005
00006 Camera::Camera() :  cameracfg(Config::getInstance().camconf){
00007     frameToggle.store(false);
00008     toggleFrame=false;
00009     cv::Mat frame(cameracfg.camResH.load(),cameracfg.camResW.load(),CV_8UC3);
00010     initialize_camera();
00011 }
00012
00013 void Camera::initialize_camera(){
00014
00015     int device = cameracfg.device.back()-'0';
00016     int width = cameracfg.camResW.load();
00017     int height = cameracfg.camResH.load();
00018     int fps = static_cast<int>(cameracfg.frameRate.load());
00019
00020     if (!cap.open(device, cv::CAP_V4L2)) {
00021         std::cerr « "Error:  Could not open camera " « device « std::endl;
00022     }
00023
00024     cap.set(cv::CAP_PROP_FRAME_WIDTH, width);
00025     cap.set(cv::CAP_PROP_FRAME_HEIGHT, height);
00026     cap.set(cv::CAP_PROP_FPS, fps);
00027
00028     int actualWidth = (int)cap.get(cv::CAP_PROP_FRAME_WIDTH);
00029     int actualHeight = (int)cap.get(cv::CAP_PROP_FRAME_HEIGHT);
00030     double actualFps = cap.get(cv::CAP_PROP_FPS);
00031
00032     if (actualWidth != width || actualHeight != height || actualFps != fps) {
00033
00034         std::cerr « "Warning:  Camera properties might not be set correctly!" « std::endl;
00035
00036         cameracfg.camResW.store(actualWidth);
```

```
00037         cameracfg.camResH.store(actualHeight);
00038         cameracfg.frameRate.store(actualFps);
00039     }
00040 }
00041
00042 Camera::~Camera(){
00043     frame.release();
00044     cap.release();
00045     std::cout«"Camera object destructed"«std::endl;
00046 }
00047
00048 bool Camera::frame_elapsed(){
00049     atomicChange = frameToggle.load();
00050     if (frameToggle.load()!=toggleFrame){     // process the current input from camera
00051         toggleFrame=atomicChange;
00052         return true;
00053     }else
00054         return false;
00055 }
00056
00057 bool Camera::capture(cv::Mat& frame){
00058
00059     cap.read(frame);
00060
00061     if(!(frame.empty())){
00062         frameToggle.store(!frameToggle.load());
00063         return true;
00064     }
00065     return false;
00066 }
```

## 7.56 src/config.cpp File Reference

```
#include <iostream>
#include "config.h"
#include "settings.h"
```

Include dependency graph for config.cpp:




## 7.57 config.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include "config.h"
00003 #include "settings.h"
```

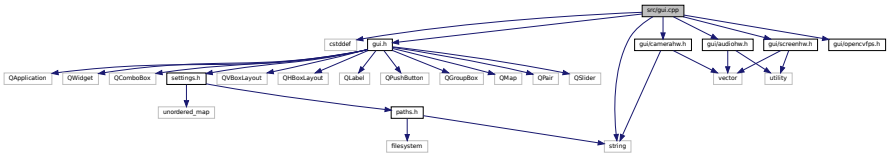
```
00004
00005 Config::Config(){
00006
00007     SettingsDB settingsDB;
00008     settings = settingsDB.loadSettings();
00009
00010     if (!settings.empty()){
00011         // Initialize audio configuration
00012         audconf.audioDevice = settings["audioDevice"];
00013         audconf.sampleRate.store(std::stoi(settings["sampleRate"])); // --> WILL BE ADJUSTED BY JACK
00014         audconf.quantization = std::stoi(settings["quantizationRatio"]);
00015         audconf.bufferSize.store(std::stoi(settings["bufferSize"])); // --> WILL BE ADJUSTED BY JACK
00016         audconf.numChannels.store(std::stoi(settings["numChannels"]));
00017         audconf.numChannels.store(std::min(audconf.numChannels.load(),2u)); // --> make numChannels
      mono or stereo and discard greater values
00018
00019
00020         // Initialize camera configuration
00021         camconf.device = settings["cameraDevice"];
00022         std::string camRes = settings["cameraResolution"];
00023         std::string temp = camRes.substr(0, camRes.find('x'));
00024         camconf.camResW.store(std::stoi(temp)); // --> WILL BE ADJUSTED BY OPENCV
00025         camconf.camResH.store(std::stoi(camRes.substr(temp.length() + 1)));  // --> WILL BE ADJUSTED
      BY OPENCV
00026         camconf.frameRate.store(std::stod(settings["cameraFrameRate"]));
00027
00028         // Initialize screen configuration
00029         std::string screenRes = settings["displayResolution"];
00030         temp = screenRes.substr(0, screenRes.find('x'));
00031         dispconf.dispResW.store(std::stoi(temp));   // --> WILL BE ADJUSTED BY OPENCV
00032         dispconf.dispResH.store(std::stoi(screenRes.substr(temp.length() + 1)));    // --> WILL BE
      ADJUSTED BY OPENCV
00033
00034         // Initialize iav configuration
00035         dispconf.fps.store(std::stod(settings["cameraFrameRate"]));
00036         if (settings["frequencyRange"] == "Narrow")
00037         {
00038             iavconf.minFrequency = 300;
00039             iavconf.maxFrequency = 700;
00040         }
00041         else if (settings["frequencyRange"] == "Normal")
00042         {
00043             iavconf.minFrequency = 300;
00044             iavconf.maxFrequency = 1500;
00045         }
00046         else if (settings["frequencyRange"] == "Wide")
00047         {
00048             iavconf.minFrequency = 100;
00049             iavconf.maxFrequency = 3000;
00050         }
00051
00052         if (settings["roi"] == "Small")
00053             iavconf.roiRadius = static_cast<int>(0.05 * camconf.camResW);
00054         else if (settings["roi"] == "Medium")
00055             iavconf.roiRadius = static_cast<int>(0.1 * camconf.camResW);
00056         else if (settings["roi"] == "Large")
00057             iavconf.roiRadius = static_cast<int>(0.15 * camconf.camResW);
00058         iavconf.trigger = settings["trigger"];
00059         iavconf.trackingAlg = settings["trackingAlgorithm"];
00060         // iavconf.skipFramesRatio = std::stoi(settings["skipFramesRatio"]);
00061         // for defining skipFramesRatio, fps/skipFramesRatio should deduce the final value of
      frameRate and fps variables.
00062     }
00063     if (runAtomicityCheck()){
00064         std::cerr«"WARNING: Atomicity is not supported on this platform for some types"«std::endl;
00065     }
00066 }
00067
00068 void Config::display(){
00069
00070     std::cout«"########## Interactive Audio Visualizer Config ##########\n";
00071     std::cout«"------------------ audio settings ----------------------\n";
00072     std::cout«"audio device             \t:\t"«audconf.audioDevice«std::endl;
00073     std::cout«"sampling rate            \t:\t"«audconf.sampleRate.load()«" samples/sec"«std::endl;
00074     std::cout«"quantization             \t:\t"«audconf.quantization«" bits"«std::endl;
00075     std::cout«"buffer size              \t:\t"«audconf.bufferSize.load()«" samples"«std::endl;
00076     std::cout«"num output channels      \t:\t"«audconf.numChannels.load()«" "«std::endl;
00077     std::cout«"------------------ display settings ---------------------\n";
00078     std::cout«"frames per second        \t:\t"«dispconf.fps.load()«" fps"«std::endl;
00079     std::cout«"display Width            \t:\t"«dispconf.dispResW.load()«" pixels"«std::endl;
00080     std::cout«"display Height           \t:\t"«dispconf.dispResH.load()«" pixels"«std::endl;
00081     std::cout«"------------------ camera settings ---------------------\n";
00082     std::cout«"camera device            \t:\t"«camconf.device«std::endl;
00083     std::cout«"camera resolution width  \t:\t"«camconf.camResW.load()«" pixels"«std::endl;
00084     std::cout«"camera resolution height \t:\t"«camconf.camResH.load()«" pixels"«std::endl;
00085     std::cout«"camera frame rate        \t:\t"«camconf.frameRate.load()«" fps"«std::endl;
00086     std::cout«"------------------ iav Settings -------------------------\n";
```

```
00087     std::cout«"mininum frequency        \t:\t"«iavconf.minFrequency«" Hz"«std::endl;
00088     std::cout«"maxinum frequency        \t:\t"«iavconf.maxFrequency«" Hz"«std::endl;
00089     std::cout«"radius                   \t:\t"«iavconf.roiRadius«" pixels"«std::endl;
00090     std::cout«"trigger method           \t:\t"«iavconf.trigger«std::endl;
00091     std::cout«"tracking algorithm       \t:\t"«iavconf.trackingAlg«std::endl;
00092     // std::cout«"skip frames ratio        \t:\t"«iavconf.skipFramesRatio«std::endl;
00093     // std::cout«"number of skip frames     \t:\t"«iavconf.skipFramesRatio-1«" frames"«std::endl;
00094     std::cout«"############################################################\n\n";
00095
00096 }
00097
00098 bool Config::runAtomicityCheck() {
00099
00100     bool warning = false;
00101
00102     // Check for atomic lock freedom for each atomic member in ConfigStruct
00103     if (!std::atomic<int>::is_always_lock_free) {
00104         std::cout « "Warning:  atomic<int> is not lock-free!\n";
00105         warning = true;
00106     }
00107
00108     if (!std::atomic<unsigned int>::is_always_lock_free) {
00109         std::cout « "Warning:  atomic<unsigned int> is not lock-free!\n";
00110         warning = true;
00111     }
00112
00113     if (!std::atomic<double>::is_always_lock_free) {
00114         std::cout « "Warning:  atomic<double> is not lock-free!\n";
00115         warning = true;
00116     }
00117
00118     return warning;
00119 }
00120
```

## 7.58 src/gui.cpp File Reference

```
#include <cstddef>
#include <string>
#include "gui.h"
#include "gui/opencvfps.h"
#include "gui/audiohw.h"
#include "gui/camerahw.h"
#include "gui/screenhw.h"
```

Include dependency graph for gui.cpp:




## 7.59 gui.cpp

Go to the documentation of this file.

```
00001 #include <cstddef>
00002 #include <string>
00003 #include "gui.h"
00004 #include "gui/opencvfps.h"
00005 #include "gui/audiohw.h"
00006 #include "gui/camerahw.h"
00007 #include "gui/screenhw.h"
00008
00009 void GUI::initializeComponents(){
00010
00011     deviceComboBox = new QComboBox();
00012     sampleRateComboBox = new QComboBox();
00013     cameraDeviceComboBox = new QComboBox();
00014     resolutionComboBox = new QComboBox();
00015     bufferSizeComboBox = new QComboBox();
00016     quantizationComboBox = new QComboBox();
```

```
00017     frameRateComboBox= new QComboBox();
00018     displayResolutionComboBox= new QComboBox();
00019     displayFrameRateComboBox= new QComboBox();
00020     frequencyRangeComboBox= new QComboBox();
00021     roiComboBox= new QComboBox();
00022     triggerComboBox= new QComboBox();
00023     trackingAlgorithmComboBox= new QComboBox();
00024     skipFramesSlider = new QSlider(Qt::Horizontal);  // Horizontal slider
00025
00026     audioDeviceLabel = new QLabel("Device:");
00027     sampleRateLabel = new QLabel("Sample Rate:");
00028     cameraDeviceLabel = new QLabel("Device:");
00029     cameraResolutionLabel = new QLabel("Resolution:");
00030     bufferSizeLabel = new QLabel("Buffer Size");
00031     quantizationLabel = new QLabel("Quantization");
00032     numOutputChannelsLabel = new QLabel("Output channels");
00033     numOutputChannelsValue = new QLabel();
00034     cameraFrameRateLabel = new QLabel("Frame Rate");
00035     screenResolutionLabel= new QLabel("Resolution");
00036     screenFrameRateLabel= new QLabel("Frames per second");
00037     iavFrequencyRangeLabel= new QLabel("Frequency range");
00038     iavRegionOfInterestLabel= new QLabel("ROI");
00039     iavTriggerLabel= new QLabel("Trigger");
00040     iavTrackingAlgLabel= new QLabel("Tracking algorithm");
00041     accuracyLabel = new QLabel("Accuracy");
00042     cpuLoadLabel = new QLabel("Economy");
00043
00044     audioDeviceLabel->setToolTip("Choose audio output device.");
00045     sampleRateLabel->setToolTip("Set samples per second.");
00046     cameraDeviceLabel->setToolTip("Select camera input device.");
00047     cameraResolutionLabel->setToolTip("Set camera capture resolution.");
00048     bufferSizeLabel->setToolTip("Configure audio buffer size.");
00049     quantizationLabel->setToolTip("Quantization range for digitalizing audio data");
00050     numOutputChannelsLabel->setToolTip("Number of output channels (>2 is Stereo)");
00051     cameraFrameRateLabel->setToolTip("Set camera frame rate (currently fixed).");
00052     screenResolutionLabel->setToolTip("Select screen resolution.");
00053     screenFrameRateLabel->setToolTip("Set screen frame rate (currently fixed).");
00054     iavFrequencyRangeLabel->setToolTip("Define audio frequency range for sound generation");
00055     iavRegionOfInterestLabel->setToolTip("Set region of interest size (window size for photshooting
      pattern).");
00056     iavTriggerLabel->setToolTip("Choose capturing method.");
00057     iavTrackingAlgLabel->setToolTip("Select object tracking algorithm.");
00058     accuracyLabel->setToolTip("Do not skip frames.");
00059     cpuLoadLabel->setToolTip("Skip frames.");
00060     skipFramesSlider->setToolTip("CURRENTLY UNSUPPORTED");
00061
00062     // create a layout for the #Outchannels
00063     numChannelsLayout = new QHBoxLayout();
00064     numChannelsLayout->addWidget(numOutputChannelsLabel);
00065     numChannelsLayout->addWidget(numOutputChannelsValue);
00066 }
00067
00068
00069 void GUI::addExplanations(){
00070
00071     // Add explanation to the audio devices,
00072     for (size_t i = 0; i<audioExplanations.size(); ++i){
00073         deviceComboBox->setItemData(i, QString::fromStdString( audioExplanations[i] ),
      Qt::ToolTipRole);
00074     }
00075
00076     // ..  to the frame rates of both camera and screen,
00077     approxFps = getCVfps_approx(cameraDeviceComboBox->currentText().toStdString().c_str());
00078     frameRateComboBox->setItemData(0, ""+ QString::number( approxFps )+" fps detected hardware
      capability", Qt::ToolTipRole);
00079     displayFrameRateComboBox->setItemData(0, ""+ QString::number( approxFps )+" fps detected hardware
      capability", Qt::ToolTipRole);
00080
00081     // ...  to the roi comboBox options,
00082     roiComboBox->setItemData(0, "5% of camera's capture resolution.", Qt::ToolTipRole);
00083     roiComboBox->setItemData(1, "10% of camera's capture resolution.", Qt::ToolTipRole);
00084     roiComboBox->setItemData(2, "15% of camera's capture resolution.", Qt::ToolTipRole);
00085
00086     // ..  to the capturing method,
00087     // triggerComboBox->setItemData(0, "Capturing is initialized manually, using the Space Bar key",
      Qt::ToolTipRole);
00088     triggerComboBox->setItemData(1, "A 5-seconds timer will initialize the capturing",
      Qt::ToolTipRole);
00089
00090     // ..  to the frequency ranges
00091     frequencyRangeComboBox->setItemData(0, "300 Hz up to 700 Hz", Qt::ToolTipRole);
00092     frequencyRangeComboBox->setItemData(1, "300 Hz up to 1500 Hz", Qt::ToolTipRole);
00093     frequencyRangeComboBox->setItemData(2, "100 Hz up to 20 kHz", Qt::ToolTipRole);
00094
00095     // ..  and finally to the tracking algorithms
00096     trackingAlgorithmComboBox->setItemData(0, "Recommended", Qt::ToolTipRole);
00097     trackingAlgorithmComboBox->setItemData(1, "Not recommended", Qt::ToolTipRole);
```

```
00098
00099 }
00100
00101 GUI::GUI() {
00102     int argc = 0;
00103     applicationStart = false;
00104
00105     QApplication app(argc,nullptr);
00106
00107     QWidget window;
00108     window.setWindowTitle("Interactive Audio Visualizer");
00109
00110     QVBoxLayout mainLayout;
00111
00112     initializeComponents();
00113
00114     initializeTexts();
00115
00116     // Audio Settings
00117     QGroupBox audioSettings("Audio Settings");
00118     QVBoxLayout audioLayout;
00119     deviceComboBox->addItems(audioDevices);
00120     audioLayout.addWidget(audioDeviceLabel);
00121     audioLayout.addWidget(deviceComboBox);
00122
00123     // Sample Rate ComboBox
00124     updateSampleRates(deviceComboBox->currentText());
00125     audioLayout.addWidget(sampleRateLabel);
00126     audioLayout.addWidget(sampleRateComboBox);
00127
00128     QObject::connect(deviceComboBox, &QComboBox::currentTextChanged,
00129     [this](const QString &text){
00130         updateSampleRates(text);
00131         updateNumChannelsInfo(text);
00132     });
00133
00134     audioLayout.addWidget(createDropDownList(bufferSizeComboBox,bufferSizeLabel, {"32", "64", "128",
      "256", "512", "1024", "2048", "4096"}));
00135     audioLayout.addWidget(createDropDownList(quantizationComboBox,quantizationLabel, {QString::number(
      AudioHardware::quantizationRatio )} ));
00136     audioLayout.addLayout(numChannelsLayout);
00137     audioSettings.setLayout(&audioLayout);
00138     mainLayout.addWidget(&audioSettings);
00139
00140     // Camera Settings
00141     QGroupBox cameraSettings("Camera Settings");
00142     QVBoxLayout cameraLayout;
00143
00144     // Camera Device ComboBox
00145     cameraDeviceComboBox->addItems(cameraDevices);
00146     cameraLayout.addWidget(cameraDeviceLabel);
00147     cameraLayout.addWidget(cameraDeviceComboBox);
00148
00149     // Resolution ComboBox
00150     updateResolution(cameraDeviceComboBox->currentText());
00151     cameraLayout.addWidget(cameraResolutionLabel);
00152     cameraLayout.addWidget(resolutionComboBox);
00153
00154     QObject::connect(cameraDeviceComboBox, &QComboBox::currentTextChanged,
00155         [this](const QString &text){
00156             updateResolution(text);
00157         });
00158
00159     cameraLayout.addWidget(createDropDownList(frameRateComboBox,cameraFrameRateLabel, {"Auto"}));
00160     cameraSettings.setLayout(&cameraLayout);
00161     mainLayout.addWidget(&cameraSettings);
00162
00163     // Display Settings
00164     QGroupBox displaySettings("Display Settings");
00165     QVBoxLayout displayLayout;
00166     displayLayout.addWidget(createDropDownList(displayResolutionComboBox,screenResolutionLabel,
      displayResolutions));
00167     displayLayout.addWidget(createDropDownList(displayFrameRateComboBox,screenFrameRateLabel,
      {"Auto"}));
00168     displaySettings.setLayout(&displayLayout);
00169     mainLayout.addWidget(&displaySettings);
00170
00171     // IAV Settings
00172     QGroupBox iavSettings("IAV Settings");
00173     QVBoxLayout iavLayout;
00174     iavLayout.addWidget(createDropDownList(frequencyRangeComboBox,iavFrequencyRangeLabel, {"Narrow",
      "Normal", "Wide"}));
00175     iavLayout.addWidget(createDropDownList(roiComboBox,iavRegionOfInterestLabel,
      {"Small","Medium","Large"}));
00176     iavLayout.addWidget(createDropDownList(triggerComboBox,iavTriggerLabel, {"Auto"})); //"Manual",
00177     iavLayout.addWidget(createDropDownList(trackingAlgorithmComboBox,iavTrackingAlgLabel, {"CSRT",
      "KCF"}));
```

```
00178     iavLayout.addWidget(createSkipFramesSlider(accuracyLabel, cpuLoadLabel));
00179     iavSettings.setLayout(&iavLayout);
00180     mainLayout.addWidget(&iavSettings);
00181 
00182     addExplanations();
00183 
00184     // Initialize errorLabel
00185     errorLabel = new QLabel();
00186     errorLabel->setStyleSheet("color:  red;");
00187     errorLabel->setText("Camera resolution cannot exceed display resolution.");
00188     errorLabel->hide(); // Initially hidden
00189     mainLayout.addWidget(errorLabel);
00190 
00191     QPushButton startButton("Start");
00192         QObject::connect(&startButton, &QPushButton::clicked, [this]() {
00193             if(checkResolutionCompatibility()) {
00194                 errorLabel->hide(); // Hide if resolutions are compatible
00195                 saveCurrentStates();
00196                 applicationStart = true;
00197                 QApplication::quit();
00198                 // exiting to start_iav();
00199             } else {
00200                 errorLabel->show(); // Show error message if not compatible
00201             }
00202         });
00203 
00204     mainLayout.addWidget(&startButton);
00205 
00206     loadCurrentStates();
00207 
00208     window.setLayout(&mainLayout);
00209     window.show();
00210 
00211     QApplication::exec();
00212 }
00213 
00214 void GUI::initializeTexts(){
00215 
00216     // Get audio devices and sample rates supported
00217     std::vector<AudioHardware::Info> audio_hw_info;
00218     get_audio_hardware_info(audio_hw_info);
00219     for (const auto&[info,sr,nChannels]:audio_hw_info){
00220         // printf("%s :  %d, %d\n",name.c_str(), sr.first, sr.second);
00221         QString audio_device = QString::fromStdString(info.first);
00222         audioExplanations.push_back(info.second);
00223         audioDevices.append(audio_device);
00224         numChannels.append(QString::number(nChannels));
00225         for (auto srate:  AudioHardware::supportedRates){
00226             if (srate >= sr.first && srate <= sr.second){
00227                 sampleRates[audio_device].append(QString::number(srate));
00228             }
00229         }
00230     }
00231 
00232     // Get available cameras and resolutions supported for each one of them
00233     auto cameras = getAvailableCameras();
00234     for (const auto& camera :  cameras) {
00235 
00236         QString camera_device = QString::fromStdString(camera.devicePath);
00237 
00238         if (!camera.resolutions.empty()){
00239             cameraDevices.append(camera_device);
00240             for (const auto& res :  camera.resolutions) {
00241                 cameraResolutions[camera_device].append(QString::number(res.first) + "x" +
      QString::number(res.second));
00242             }
00243         }
00244     }
00245 
00246     // Get screen resolutions for the main screen
00247     auto screen_resolutions = get_screen_resolution();
00248     for(const auto&resolution :  screen_resolutions){
00249         displayResolutions.append(QString::number(resolution.first) + "x" +
      QString::number(resolution.second));
00250     }
00251 
00252 }
00253 
00254 void GUI::saveCurrentStates(){
00255 
00256     std::unordered_map<std::string, std::string> settings;
00257 
00258     settings["audioDevice"] = deviceComboBox->currentText().toStdString();
00259     settings["sampleRate"] = sampleRateComboBox->currentText().toStdString();
00260     settings["cameraDevice"] = cameraDeviceComboBox->currentText().toStdString();
00261     settings["cameraResolution"] = resolutionComboBox->currentText().toStdString();
00262     settings["cameraFrameRate"] = std::to_string(approxFps);
```

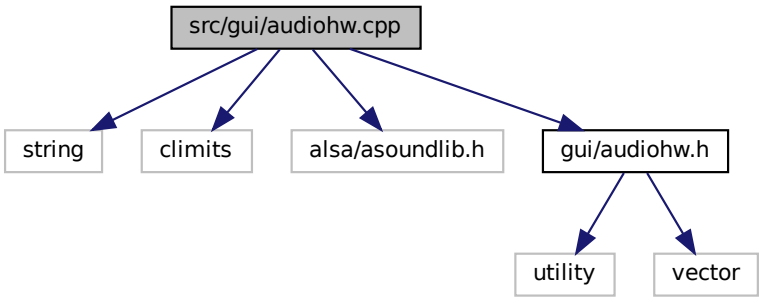
```
00263     settings["bufferSize"] = bufferSizeComboBox->currentText().toStdString();
00264     settings["quantization"] = quantizationComboBox->currentText().toStdString();
00265     settings["numChannels"] = numOutputChannelsValue->text().toStdString();
00266     settings["quantizationRatio"] = std::to_string(AudioHardware::quantizationRatio);
00267     settings["frameRate"] = frameRateComboBox->currentText().toStdString();
00268     settings["displayResolution"] = displayResolutionComboBox->currentText().toStdString();
00269     settings["displayFrameRate"] = displayFrameRateComboBox->currentText().toStdString();
00270     settings["frequencyRange"] = frequencyRangeComboBox->currentText().toStdString();
00271     settings["roi"] = roiComboBox->currentText().toStdString();
00272     settings["trigger"] = triggerComboBox->currentText().toStdString();
00273     settings["trackingAlgorithm"] = trackingAlgorithmComboBox->currentText().toStdString();
00274     settings["skipFramesRatio"] = std::to_string(skipFramesSlider->value());
00275 
00276     settingsDB.saveSettings(settings);
00277 }
00278 
00279 void GUI::loadCurrentStates(){
00280 
00281     auto settings = settingsDB.loadSettings();
00282 
00283     if (!settings.empty()){
00284 
00285         if (audioDevices.contains(QString::fromStdString(settings["audioDevice"]))){
00286             deviceComboBox->setCurrentText(QString::fromStdString(settings["audioDevice"]));
00287         }
00288         if
(sampleRates[QString::fromStdString(settings["audioDevice"])].contains(QString::fromStdString(settings["sampleRate"])))
00289             sampleRateComboBox->setCurrentText(QString::fromStdString(settings["sampleRate"]));
00290         }
00291         if (cameraDevices.contains(QString::fromStdString(settings["cameraDevice"]))){
00292             cameraDeviceComboBox->setCurrentText(QString::fromStdString(settings["cameraDevice"]));
00293         }
00294         if
(cameraResolutions[QString::fromStdString(settings["cameraDevice"])].contains(QString::fromStdString(settings["cameraRes
00295             resolutionComboBox->setCurrentText(QString::fromStdString(settings["cameraResolution"]));
00296         }
00297         bufferSizeComboBox->setCurrentText(QString::fromStdString(settings["bufferSize"]));
00298         // numOutputChannelsValue->setText(QString::fromStdString(settings["numChannels"]));
00299         frameRateComboBox->setCurrentText(QString::fromStdString(settings["frameRate"]));
00300         if (displayResolutions.contains(QString::fromStdString(settings["displayResolution"]))){
00301 
displayResolutionComboBox->setCurrentText(QString::fromStdString(settings["displayResolution"]));
00302         }
00303 
displayFrameRateComboBox->setCurrentText(QString::fromStdString(settings["displayFrameRate"]));
00304         frequencyRangeComboBox->setCurrentText(QString::fromStdString(settings["frequencyRange"]));
00305         roiComboBox->setCurrentText(QString::fromStdString(settings["roi"]));
00306         triggerComboBox->setCurrentText(QString::fromStdString(settings["trigger"]));
00307 
trackingAlgorithmComboBox->setCurrentText(QString::fromStdString(settings["trackingAlgorithm"]));
00308         skipFramesSlider->setValue(std::stoi(settings["skipFramesRatio"]));
00309     }
00310 }
00311 
00312 void GUI::updateSampleRates(const QString &audioDevice) {
00313     sampleRateComboBox->clear();
00314     sampleRateComboBox->addItems(sampleRates[audioDevice]);
00315 }
00316 
00317 // static int lala = 1;
00318 void GUI::updateNumChannelsInfo(const QString &audioDevice) {
00319     numOutputChannelsValue->setText(numChannels[audioDevices.indexOf(audioDevice)]);
00320 }
00321 
00322 void GUI::updateResolution(const QString &cameraDevice) {
00323     resolutionComboBox->clear();
00324     resolutionComboBox->addItems(cameraResolutions[cameraDevice]);
00325 }
00326 
00327 QWidget* GUI::createDropDownList(QComboBox *comboBox,QLabel *label,const QStringList& comboBoxItems) {
00328 
00329     auto *rowLayout = new QHBoxLayout;
00330     comboBox->addItems(comboBoxItems);
00331 
00332     rowLayout->addWidget(label);
00333     rowLayout->addWidget(comboBox);
00334 
00335     auto *rowWidget = new QWidget;
00336     rowWidget->setLayout(rowLayout);
00337 
00338     return rowWidget;
00339 }
00340 
00341 QWidget* GUI::createSkipFramesSlider(QLabel *label1, QLabel *label2) {
00342 
00343     skipFramesSlider->setMinimum(1);
00344     skipFramesSlider->setMaximum(5);
```

```
00345     skipFramesSlider->setValue(0);
00346
00347     QWidget* sliderWidget = new QWidget();
00348     QHBoxLayout* layout = new QHBoxLayout(sliderWidget);
00349
00350     layout->addWidget(label1, 0, Qt::AlignLeft);
00351     layout->addWidget(skipFramesSlider);
00352     layout->addWidget(label2, 0, Qt::AlignRight);
00353
00354     // optional ..
00355     // connect(skipFramesSlider, &QSlider::valueChanged, this, &Gui::onSliderValueChanged);
00356
00357     sliderWidget->setLayout(layout);  // Set the layout to the container widget
00358     return sliderWidget;  // Return the widget containing the slider and labels
00359 }
00360
00361 bool GUI::checkResolutionCompatibility() {
00362     // Resolutions are currently stored as "WidthxHeight"
00363     QString cameraRes = resolutionComboBox->currentText();
00364     QString displayRes = displayResolutionComboBox->currentText();
00365
00366     int cameraWidth = cameraRes.split("x")[0].toInt();
00367     int cameraHeight = cameraRes.split("x")[1].toInt();
00368     int displayWidth = displayRes.split("x")[0].toInt();
00369     int displayHeight = displayRes.split("x")[1].toInt();
00370
00371     return (cameraWidth <= displayWidth && cameraHeight <= displayHeight);
00372 }
00373
00374 bool GUI::onExit(){
00375     return !applicationStart;
00376 }
```

## 7.60 src/gui/audiohw.cpp File Reference

```
#include <string>
#include <climits>
#include <alsa/asoundlib.h>
#include "gui/audiohw.h"
```

Include dependency graph for audiohw.cpp:




## 7.61 audiohw.cpp

Go to the documentation of this file.

```
00001 #include <string>
00002 #include <climits>
00003 #include <alsa/asoundlib.h>
00004 #include "gui/audiohw.h"
00005
00006 // using namespace AudioHardware;
00007
00008 bool AudioHardware::get_audio_device_info(int card, int device, std::pair<unsigned int, unsigned int>
      &sample_rate,unsigned int &numChannels)
00009 {
```

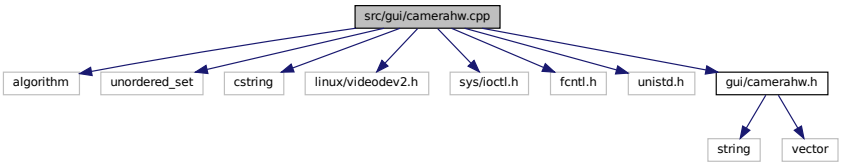
```
00010     snd_pcm_t *handle;
00011     snd_pcm_hw_params_t *params;
00012     int err;
00013     char name[32];
00014     unsigned int sample_rate_min,
00015                  sample_rate_max;
00016
00017     // Open the PCM device
00018     sprintf(name, "hw:%d,%d", card, device);
00019     err = snd_pcm_open(&handle, name, SND_PCM_STREAM_PLAYBACK, 0);
00020     if (err < 0) {
00021         // Error opening PCM device
00022         return false;
00023     }
00024
00025     // Allocate hardware parameters object
00026     snd_pcm_hw_params_alloca(&params);
00027
00028     // Initialize hwparams with full configuration space
00029     err = snd_pcm_hw_params_any(handle, params);
00030     if (err < 0) {
00031         // Error setting hwparams
00032         snd_pcm_close(handle);
00033         return false;
00034     }
00035
00036     // Get sample rate range
00037     err = snd_pcm_hw_params_get_rate_min(params, &sample_rate_min, nullptr);
00038     if (err < 0) {
00039         // Error getting sample rate min
00040         snd_pcm_close(handle);
00041         return false;
00042     }
00043     err = snd_pcm_hw_params_get_rate_max(params, &sample_rate_max, nullptr);
00044     if (err < 0) {
00045         // Error getting sample rate max
00046         snd_pcm_close(handle);
00047         return false;
00048     }
00049
00050     sample_rate.first = sample_rate_min;
00051     sample_rate.second = sample_rate_max;
00052
00053     // get number of output channels
00054     err = snd_pcm_hw_params_get_channels(params, &numChannels);// channels now holds the number of
      channels (outputs)
00055     if (err < 0 || numChannels ==0 ) {
00056         // Set the desired number of channels (e.g., 2 for stereo)
00057         unsigned int atLeastStereo = 2;
00058         err = snd_pcm_hw_params_set_channels(handle, params, atLeastStereo);
00059         if (err < 0 || numChannels ==0 ) {
00060
00061             unsigned int atLeastMono = 1;
00062             err = snd_pcm_hw_params_set_channels(handle, params, atLeastMono);
00063             if (err < 0 || numChannels ==0) {
00064                 // Error setting channels
00065                 snd_pcm_close(handle);
00066                 return false;
00067             }
00068             // set numChannels to mono
00069             snd_pcm_hw_params_get_channels(params, &numChannels);
00070         } else {
00071             //set numChannels to stereo
00072             snd_pcm_hw_params_get_channels(params, &numChannels);
00073         }
00074     }
00075
00076     // Close the PCM device
00077     snd_pcm_close(handle);
00078     return true;
00079 }
00080
00081 void AudioHardware::get_audio_hardware_info(std::vector<Info> &audio_hw_info){
00082
00083     int card = -1;
00084
00085     // Loop through all available cards
00086     while (true) {
00087
00088         // Find the next card
00089         int err = snd_card_next(&card);
00090         if (err < 0) {
00091             // Error getting next card
00092             break;
00093         }
00094         if (card < 0) {
00095             // No more cards
```

```
00096             break;
00097         }
00098
00099         // Open the card control interface
00100         snd_ctl_t *ctl_handle;
00101         char ctl_name[32];
00102         sprintf(ctl_name, "hw:%d", card);
00103         err = snd_ctl_open(&ctl_handle, ctl_name, 0);
00104         if (err < 0) {
00105             // Error opening card
00106             continue;
00107         }
00108
00109         // Get the card info
00110         snd_ctl_card_info_t *info;
00111         snd_ctl_card_info_malloc(&info);
00112         err = snd_ctl_card_info(ctl_handle, info);
00113         if (err < 0) {
00114             // Error getting card info
00115             snd_ctl_close(ctl_handle);
00116             snd_ctl_card_info_free(info);
00117             continue;
00118         }
00119
00120         std::string card_id = snd_ctl_card_info_get_id(info) ;
00121         std::string mixer = snd_ctl_card_info_get_mixername(info);
00122 // card_id = card_id + "("+mixer+")";
00123
00124         // Check if it is an output device and if it can be opened
00125         snd_pcm_t *handle;
00126         std::string card_name_str ="hw:" + std::to_string(card) + ",0";
00127         const char* card_name = card_name_str.c_str();
00128         if (snd_pcm_open(&handle, card_name, SND_PCM_STREAM_PLAYBACK, 0) >= 0) {
00129             snd_pcm_close(handle);
00130         } else {
00131             continue; // Skip this card if it doesn't support output.
00132         }
00133
00134         // Free the card info
00135         snd_ctl_card_info_free(info);
00136         snd_ctl_close(ctl_handle);
00137
00138         // Get PCM device info
00139         int device = 0;
00140         unsigned int numChannels = 0;
00141         std::pair<unsigned int, unsigned int> sample_rate_range;
00142         while(!AudioHardware::get_audio_device_info(card, device,sample_rate_range,numChannels) &&
      device < MAX_POTENTIAL_AUDIO_DEVICES ){
00143             ++device;
00144         }
00145
00146         Info deviceInfo;
00147         deviceInfo.card_info = std::make_pair(card_id,mixer);
00148         deviceInfo.sample_rate_range = sample_rate_range;
00149         deviceInfo.numberOfChannels = numChannels;
00150         audio_hw_info.push_back(deviceInfo);
00151
00152     }
00153 }
```

## 7.62 src/gui/camerahw.cpp File Reference

```
#include <algorithm>
#include <unordered_set>
#include <cstring>
#include <linux/videodev2.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <unistd.h>
#include "gui/camerahw.h"
```

Include dependency graph for camerahw.cpp:




## Functions

- std::vector< CameraInfo > getAvailableCameras ()

  *Retrieves information about available cameras on the system.*

## 7.62.1 Function Documentation

### 7.62.1.1 getAvailableCameras()

`std::vector< CameraInfo > getAvailableCameras ( )`

Retrieves information about available cameras on the system.

This function scans the system for connected cameras and gathers information about each camera, including its device path and supported resolutions.

**Returns**

A vector of CameraInfo structures, where each structure contains information about a single camera. If no cameras are found, an empty vector is returned.

Definition at line 15 of file camerahw.cpp.

```
00015                                                {
00016     std::vector<CameraInfo> cameras;
00017     std::unordered_set<std::string> uniquesResolutionValues;
00018 
00019     for (int i = 0; i < 16; ++i) { // Check up to 16 potential camera devices
00020         std::string devicePath = "/dev/video" + std::to_string(i);
00021         int fd = open(devicePath.c_str(), O_RDWR | O_NONBLOCK, 0);
00022 
00023         if (fd == -1) {
00024             if (errno == ENOENT || errno == EACCES) {
00025                 continue; // Device doesn't exist or no permission, try next
00026             } else {
00027                 perror("open");
00028                 continue; // Some other error, try next
00029             }
00030         }
00031 
00032 
00033         struct v4l2_capability cap;
00034         if (ioctl(fd, VIDIOC_QUERYCAP, &cap) == -1) {
00035             perror("VIDIOC_QUERYCAP");
00036             close(fd);
00037             continue;
00038         }
00039 
00040         if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
00041              //Not a video capture device
00042             close(fd);
00043             continue;
00044         }
00045 
00046         CameraInfo camera;
00047         camera.devicePath = devicePath;
00048 
00049 
00050 
00051         struct v4l2_fmtdesc fmtdesc;
00052         memset(&fmtdesc, 0, sizeof(fmtdesc));
```

```
00053         fmtdesc.index = 0;
00054         fmtdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
00055 
00056         while (ioctl(fd, VIDIOC_ENUM_FMT, &fmtdesc) == 0) {
00057             struct v4l2_frmsizeenum frmsizeenum;
00058             memset(&frmsizeenum, 0, sizeof(frmsizeenum));
00059             frmsizeenum.index = 0;
00060             frmsizeenum.pixel_format = fmtdesc.pixelformat;
00061 
00062             while (ioctl(fd, VIDIOC_ENUM_FRAMESIZES, &frmsizeenum) == 0) {
00063 
00064                 if (frmsizeenum.type == V4L2_FRMSIZE_TYPE_DISCRETE) {
00065                     std::string resVal = std::to_string(frmsizeenum.discrete.width) +"x"+
      std::to_string(frmsizeenum.discrete.height);
00066                     if(uniquesResolutionValues.find(resVal)== uniquesResolutionValues.end()){
00067                         camera.resolutions.push_back({frmsizeenum.discrete.width,
      frmsizeenum.discrete.height});
00068                         uniquesResolutionValues.insert(resVal);
00069                     }
00070                 } else if (frmsizeenum.type == V4L2_FRMSIZE_TYPE_STEPWISE) {
00071                      for (size_t w = frmsizeenum.stepwise.min_width; w <=
      frmsizeenum.stepwise.max_width; w += frmsizeenum.stepwise.step_width) {
00072                         for (size_t h = frmsizeenum.stepwise.min_height; h <=
      frmsizeenum.stepwise.max_height; h += frmsizeenum.stepwise.step_height) {
00073 
00074                             std::string resVal = std::to_string(w) +"x"+ std::to_string(h);
00075                             if(uniquesResolutionValues.find(resVal)== uniquesResolutionValues.end()){
00076                                 camera.resolutions.push_back({w,h});
00077                                 uniquesResolutionValues.insert(resVal);
00078                             }
00079                         }
00080                     }
00081                 }
00082 
00083                 frmsizeenum.index++;
00084             }
00085              fmtdesc.index++;
00086         }
00087 
00088         std::sort(camera.resolutions.begin(),camera.resolutions.end(),compareResolutions);
00089         cameras.push_back(camera);
00090         close(fd);
00091     }
00092     return cameras;
00093 }
```

## 7.63 camerahw.cpp

Go to the documentation of this file.

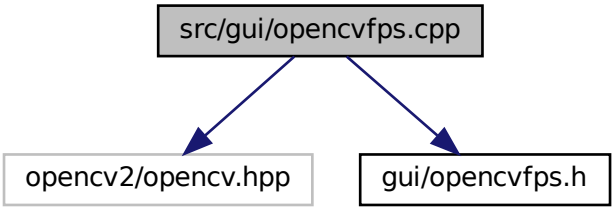
```
00001 #include <algorithm>
00002 #include <unordered_set>
00003 #include <cstring>
00004 #include <linux/videodev2.h>
00005 #include <sys/ioctl.h>
00006 #include <fcntl.h>
00007 #include <unistd.h>
00008 // #include <errno.h>
00009 #include "gui/camerahw.h"
00010 
00011 static bool compareResolutions(std::pair<int, int> r1,std::pair<int, int> r2){
00012     return (r1.first > r2.first) ?  true :  (r1.second > r2.second);
00013 }
00014 
00015 std::vector<CameraInfo> getAvailableCameras() {
00016     std::vector<CameraInfo> cameras;
00017     std::unordered_set<std::string> uniquesResolutionValues;
00018 
00019     for (int i = 0; i < 16; ++i) { // Check up to 16 potential camera devices
00020         std::string devicePath = "/dev/video" + std::to_string(i);
00021         int fd = open(devicePath.c_str(), O_RDWR | O_NONBLOCK, 0);
00022 
00023         if (fd == -1) {
00024             if (errno == ENOENT || errno == EACCES) {
00025                 continue; // Device doesn't exist or no permission, try next
00026             } else {
00027                 perror("open");
00028                 continue; // Some other error, try next
00029             }
00030         }
00031 
00032 
00033         struct v4l2_capability cap;
00034         if (ioctl(fd, VIDIOC_QUERYCAP, &cap) == -1) {
00035             perror("VIDIOC_QUERYCAP");
```

```
00036             close(fd);
00037             continue;
00038         }
00039 
00040         if (!(cap.capabilities & V4L2_CAP_VIDEO_CAPTURE)) {
00041              //Not a video capture device
00042             close(fd);
00043             continue;
00044         }
00045 
00046         CameraInfo camera;
00047         camera.devicePath = devicePath;
00048 
00049 
00050 
00051         struct v4l2_fmtdesc fmtdesc;
00052         memset(&fmtdesc, 0, sizeof(fmtdesc));
00053         fmtdesc.index = 0;
00054         fmtdesc.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
00055 
00056         while (ioctl(fd, VIDIOC_ENUM_FMT, &fmtdesc) == 0) {
00057             struct v4l2_frmsizeenum frmsizeenum;
00058             memset(&frmsizeenum, 0, sizeof(frmsizeenum));
00059             frmsizeenum.index = 0;
00060             frmsizeenum.pixel_format = fmtdesc.pixelformat;
00061 
00062             while (ioctl(fd, VIDIOC_ENUM_FRAMESIZES, &frmsizeenum) == 0) {
00063 
00064                 if (frmsizeenum.type == V4L2_FRMSIZE_TYPE_DISCRETE) {
00065                     std::string resVal = std::to_string(frmsizeenum.discrete.width) +"x"+
      std::to_string(frmsizeenum.discrete.height);
00066                     if(uniquesResolutionValues.find(resVal)== uniquesResolutionValues.end()){
00067                         camera.resolutions.push_back({frmsizeenum.discrete.width,
      frmsizeenum.discrete.height});
00068                         uniquesResolutionValues.insert(resVal);
00069                     }
00070                 } else if (frmsizeenum.type == V4L2_FRMSIZE_TYPE_STEPWISE) {
00071                      for (size_t w = frmsizeenum.stepwise.min_width; w <=
      frmsizeenum.stepwise.max_width; w += frmsizeenum.stepwise.step_width) {
00072                         for (size_t h = frmsizeenum.stepwise.min_height; h <=
      frmsizeenum.stepwise.max_height; h += frmsizeenum.stepwise.step_height) {
00073 
00074                             std::string resVal = std::to_string(w) +"x"+ std::to_string(h);
00075                             if(uniquesResolutionValues.find(resVal)== uniquesResolutionValues.end()){
00076                                 camera.resolutions.push_back({w,h});
00077                                 uniquesResolutionValues.insert(resVal);
00078                             }
00079                         }
00080                     }
00081                 }
00082 
00083                 frmsizeenum.index++;
00084             }
00085              fmtdesc.index++;
00086         }
00087 
00088         std::sort(camera.resolutions.begin(),camera.resolutions.end(),compareResolutions);
00089         cameras.push_back(camera);
00090         close(fd);
00091     }
00092     return cameras;
00093 }
00094 
00095 
00096 
00097 
00098 
00099 // int main() {
00100 //     auto cameras = getAvailableCameras();
00101 
00102 //     for (const auto& camera :  cameras) {
00103 //         std::cout « "Camera:  " « camera.devicePath « std::endl;
00104 //         std::cout « "Available resolutions:" « std::endl;
00105 //         for (const auto& res :  camera.resolutions) {
00106 //             std::cout « "\t" « res.first « "x" « res.second « std::endl;
00107 //         }
00108 //         std::cout « std::endl;
00109 //     }
00110 
00111 
00112 //     return 0;
00113 // }
```

## 7.64 src/gui/opencvfps.cpp File Reference

```
#include "opencv2/opencv.hpp"
#include "gui/opencvfps.h"
```

Include dependency graph for opencvfps.cpp:




### Functions

- double getCVfps_approx (const char *cameraDevice)

  *Calculates the camera's approximate frames per second (FPS) based on OpenCV's video capture.*

### 7.64.1 Function Documentation

#### 7.64.1.1 getCVfps_approx()

```
double getCVfps_approx (
            const char * cameraDevice )
```

Calculates the camera's approximate frames per second (FPS) based on OpenCV's video capture.

**Parameters**

| | |
|---|---|
| *const* | char* cameraDevice - The camera device id. |

**Returns**

double - The approximate frames per second (FPS) calculated from the video capture.

Definition at line 4 of file opencvfps.cpp.

```
00004                                                {
00005     cv::VideoCapture video(cameraDevice);
00006
00007     // Check camera
00008     // if (!video.isOpened()) {
00009     //     return -1;
00010     // }
00011
00012     return video.get(cv::CAP_PROP_FPS);
00013 }
```

## 7.65 opencvfps.cpp

Go to the documentation of this file.

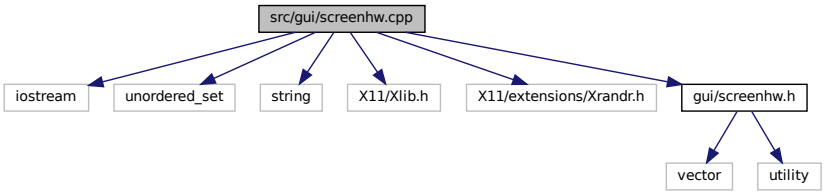
```
00001 #include "opencv2/opencv.hpp"
00002 #include "gui/opencvfps.h"
00003
```

```
00004 double getCVfps_approx(const char* cameraDevice){
00005     cv::VideoCapture video(cameraDevice);
00006 
00007     // Check camera
00008     // if (!video.isOpened()) {
00009     //     return -1;
00010     // }
00011 
00012     return video.get(cv::CAP_PROP_FPS);
00013 }
```

# 7.66 src/gui/screenhw.cpp File Reference

```
#include <iostream>
#include <unordered_set>
#include <string>
#include <X11/Xlib.h>
#include <X11/extensions/Xrandr.h>
#include "gui/screenhw.h"
```

Include dependency graph for screenhw.cpp:




## Functions

- std::vector< std::pair< int, int > > get_screen_resolution ()

  *Retrieves the resolution of the primary (default) screen using platform-specific APIs.*

## 7.66.1 Function Documentation

### 7.66.1.1 get_screen_resolution()

```
std::vector< std::pair< int, int > > get_screen_resolution ( )
```

Retrieves the resolution of the primary (default) screen using platform-specific APIs.

**Returns**

std::vector<std::pair<int,int>> - A list of pairs of integers representing the list of width and height supported by the screen.

Definition at line 10 of file screenhw.cpp.

```
00010                                                 {
00011 
00012     std::vector<std::pair<int,int>> screen_resolutions;
00013     std::unordered_set<std::string> uniquesValues;
00014 
00015     Display* display = XOpenDisplay(NULL);
00016     if (!display) {
00017         std::cerr << "Error:  Couldn't open display." << std::endl;
00018         return {};
00019     }
00020 
00021     int screen = DefaultScreen(display);
00022     Window root = RootWindow(display, screen);
00023 
```

```
00024     XRRScreenResources* resources = XRRGetScreenResourcesCurrent(display, root);
00025     if (!resources) {
00026         std::cerr « "Error:  Could not get screen resources." « std::endl;
00027         XCloseDisplay(display);
00028         return {};
00029     }
00030
00031     // std::cout « "Available Screen Resolutions:" « std::endl;
00032
00033     // Iterate through all modes and display valid resolutions
00034     for (int i = 0; i < resources->nmode; ++i) {
00035         XRRModeInfo* mode = &resources->modes[i];
00036         if (mode->width > 0 && mode->height > 0) { // Ensure valid dimensions
00037             // std::cout « mode->width « "x" « mode->height « std::endl;
00038             std::string resVal = std::to_string(mode->width) +"x"+ std::to_string(mode->height);
00039             if(uniquesValues.find(resVal)== uniquesValues.end()){
00040                 screen_resolutions.push_back({mode->width, mode->height});
00041                 uniquesValues.insert(resVal);
00042             }
00043         }
00044     }
00045
00046     XRRFreeScreenResources(resources);
00047     XCloseDisplay(display);
00048
00049     return screen_resolutions;
00050 }
```

## 7.67 screenhw.cpp

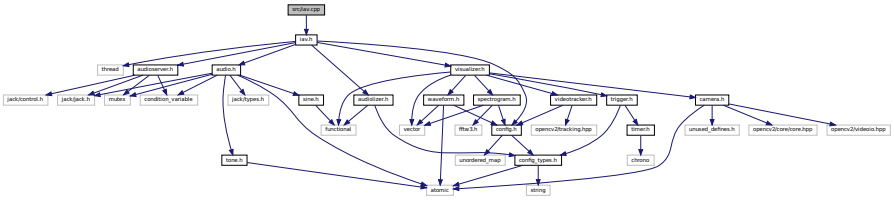
Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <unordered_set>
00003 #include <string>
00004 #include <X11/Xlib.h>
00005 #include <X11/extensions/Xrandr.h>
00006 #include "gui/screenhw.h"
00007
00008
00009
00010 std::vector<std::pair<int,int» get_screen_resolution() {
00011
00012     std::vector<std::pair<int,int» screen_resolutions;
00013     std::unordered_set<std::string> uniquesValues;
00014
00015     Display* display = XOpenDisplay(NULL);
00016     if (!display) {
00017         std::cerr « "Error:  Couldn't open display." « std::endl;
00018         return {};
00019     }
00020
00021     int screen = DefaultScreen(display);
00022     Window root = RootWindow(display, screen);
00023
00024     XRRScreenResources* resources = XRRGetScreenResourcesCurrent(display, root);
00025     if (!resources) {
00026         std::cerr « "Error:  Could not get screen resources." « std::endl;
00027         XCloseDisplay(display);
00028         return {};
00029     }
00030
00031     // std::cout « "Available Screen Resolutions:" « std::endl;
00032
00033     // Iterate through all modes and display valid resolutions
00034     for (int i = 0; i < resources->nmode; ++i) {
00035         XRRModeInfo* mode = &resources->modes[i];
00036         if (mode->width > 0 && mode->height > 0) { // Ensure valid dimensions
00037             // std::cout « mode->width « "x" « mode->height « std::endl;
00038             std::string resVal = std::to_string(mode->width) +"x"+ std::to_string(mode->height);
00039             if(uniquesValues.find(resVal)== uniquesValues.end()){
00040                 screen_resolutions.push_back({mode->width, mode->height});
00041                 uniquesValues.insert(resVal);
00042             }
00043         }
00044     }
00045
00046     XRRFreeScreenResources(resources);
00047     XCloseDisplay(display);
00048
00049     return screen_resolutions;
00050 }
```

## 7.68 src/iav.cpp File Reference

```
#include "iav.h"
```

Include dependency graph for iav.cpp:




## 7.69 iav.cpp

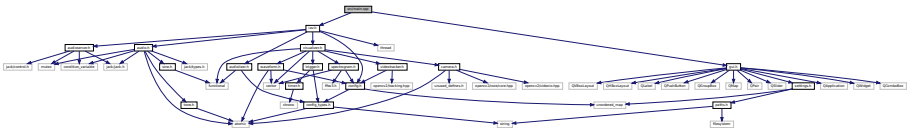
Go to the documentation of this file.

```
00001 #include "iav.h"
00002
00003 IAV::IAV()
00004 {
00005     audioServer.setup_server();
00006
00007
      audiolizer.setAudioUpdater(std::bind(&AudioStream::update,&audioStream,std::placeholders::_1,std::placeholders::_2));
00008     visualizer.setAudiolizerUpdater(std::bind(&Audiolizer::turn_Image_into_Sound, &audiolizer,
      std::placeholders::_1,std::placeholders::_2,std::placeholders::_3,std::placeholders::_4));
00009     audioStream.setVisualizerUpdater(std::bind(&Visualizer::updateAudioSignal, &visualizer,
      std::placeholders::_1));
00010
00011     audServerThread = std::thread (&AudioServer::start_server,&audioServer,std::ref(mtxServer),
      std::ref(cvServer), std::ref(serverStarted));
00012     audioThread = std::thread (&AudioStream::clientConnect,&audioStream,std::ref(mtxServer),
      std::ref(cvServer), std::ref(serverStarted));
00013     visualThread = std::thread(&Visualizer::broadcast,&visualizer);
00014
00015     cfg.display();
00016 }
00017
00018 IAV::~IAV(){
00019     audioServer.~AudioServer();
00020     audioStream.~AudioStream();
00021     audiolizer.~Audiolizer() ;
00022     visualizer.~Visualizer();
00023 }
00024
00025 void IAV::start(){
00026
00027     audServerThread.detach();
00028     audioThread.detach();
00029     visualThread.join();
00030
00031 }
```

## 7.70 src/main.cpp File Reference

```
#include "iav.h"
#include "gui.h"
```

Include dependency graph for main.cpp:



## Functions

- int main ()

  *main function.*

### 7.70.1 Detailed Description

**Author**

Melissas Paschalis melissaspaschalis@gmail.com

**Version**

2.0

### 7.70.2 LICENSE

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version.
Definition in file main.cpp.

### 7.70.3 Function Documentation

#### 7.70.3.1 main()

`int main ( )`
main function.

**Parameters**

| | |
|---|---|
| *int* | argc - number of input arguments. |
| *char* | **argv - arguments. No additional arguments are required. These variables are only used to initialize the QT application instance. |

**Returns**

int - success / failure of program

Definition at line 21 of file main.cpp.

```
00021           {
00022
00023     GUI gui;
00024     if (!gui.onExit()){
00025       IAV interactiveAudioVisualizer;
00026       interactiveAudioVisualizer.start();
00027     }
00028
00029     return 0;
00030 }
```

## 7.71 main.cpp

Go to the documentation of this file.

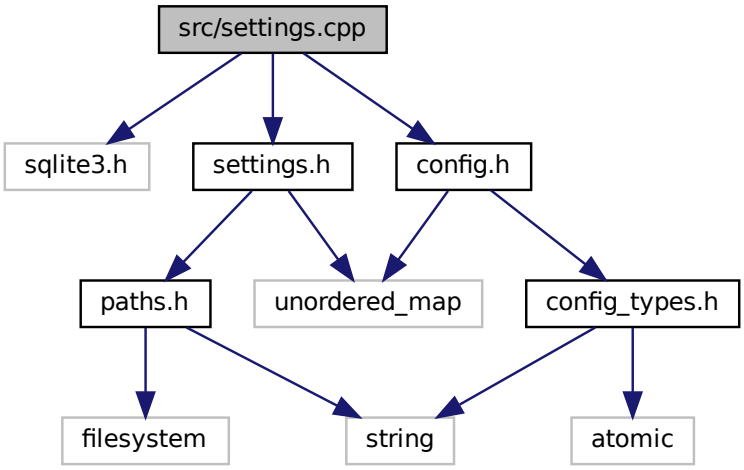
```
00001 /*!*****************************************/
00013 #include "iav.h" // iav.h before gui.h
00014 #include "gui.h"
00015
00021 int main(){
00022
00023     GUI gui;
00024     if (!gui.onExit()){
00025       IAV interactiveAudioVisualizer;
```

```
00026       interactiveAudioVisualizer.start();
00027     }
00028
00029     return 0;
00030 }
```

## 7.72 src/settings.cpp File Reference

```
#include "sqlite3.h"
#include "settings.h"
#include "config.h"
```

Include dependency graph for settings.cpp:




## 7.73 settings.cpp

Go to the documentation of this file.

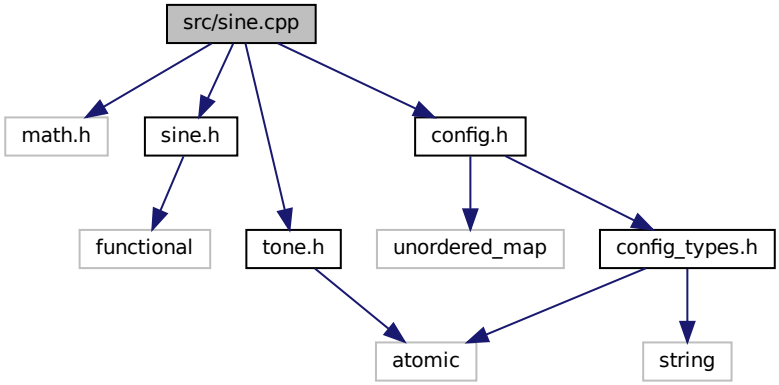
```
00001 #include "sqlite3.h"
00002 #include "settings.h"
00003 #include "config.h"
00004
00005 SettingsDB::SettingsDB(const std::string& db_path) :  dbPath(db_path) {
00006
00007     if (sqlite3_open(dbPath.c_str(), &db) != SQLITE_OK) {
00008         db = nullptr;
00009         return;
00010     }
00011
00012     // Create table if it doesn't exist
00013     const char* createTableSQL =
00014         "CREATE TABLE IF NOT EXISTS settings ("
00015         "setting_name TEXT PRIMARY KEY,"
00016         "setting_value TEXT NOT NULL);";
00017
00018     char* errMsg = nullptr;
00019     if (sqlite3_exec(db, createTableSQL, nullptr, nullptr, &errMsg) != SQLITE_OK) {
00020         sqlite3_free(errMsg);
00021         sqlite3_close(db);
00022         db = nullptr;
00023     }
00024 }
00025
00026 SettingsDB::~SettingsDB(){
00027
00028     if (db) {
00029         sqlite3_close(db);
00030     }
```

```
00031
00032 }
00033
00034 bool SettingsDB::saveSettings(const std::unordered_map<std::string, std::string>& settings) {
00035     if (!db) return false;
00036
00037     // Begin transaction for better performance
00038     char* errMsg = nullptr;
00039     if (sqlite3_exec(db, "BEGIN TRANSACTION", nullptr, nullptr, &errMsg) != SQLITE_OK) {
00040         sqlite3_free(errMsg);
00041         return false;
00042     }
00043
00044     // First, clear existing settings
00045     const char* clearSQL = "DELETE FROM settings;";
00046     if (sqlite3_exec(db, clearSQL, nullptr, nullptr, &errMsg) != SQLITE_OK) {
00047         sqlite3_free(errMsg);
00048         return false;
00049     }
00050
00051     // Prepare the insert statement
00052     sqlite3_stmt* stmt;
00053     const char* insertSQL = "INSERT INTO settings (setting_name, setting_value) VALUES (?, ?);";
00054     if (sqlite3_prepare_v2(db, insertSQL, -1, &stmt, nullptr) != SQLITE_OK) {
00055         return false;
00056     }
00057
00058     // Insert all settings
00059     for (const auto& [key, value] :  settings) {
00060         sqlite3_bind_text(stmt, 1, key.c_str(), -1, SQLITE_STATIC);
00061         sqlite3_bind_text(stmt, 2, value.c_str(), -1, SQLITE_STATIC);
00062
00063         if (sqlite3_step(stmt) != SQLITE_DONE) {
00064             sqlite3_finalize(stmt);
00065             return false;
00066         }
00067         sqlite3_reset(stmt);
00068     }
00069
00070     sqlite3_finalize(stmt);
00071
00072     if (sqlite3_exec(db, "COMMIT", nullptr, nullptr, &errMsg) != SQLITE_OK) {
00073         sqlite3_free(errMsg);
00074         return false;
00075     }
00076
00077     return true;
00078 }
00079
00080 std::unordered_map<std::string, std::string> SettingsDB::loadSettings() {
00081     std::unordered_map<std::string, std::string> settings;
00082
00083     if (!db) return settings;
00084
00085     const char* selectSQL = "SELECT setting_name, setting_value FROM settings;";
00086     sqlite3_stmt* stmt;
00087
00088     if (sqlite3_prepare_v2(db, selectSQL, -1, &stmt, nullptr) != SQLITE_OK) {
00089         return settings;
00090     }
00091
00092     while (sqlite3_step(stmt) == SQLITE_ROW) {
00093         std::string name = reinterpret_cast<const char*>(sqlite3_column_text(stmt, 0));
00094         std::string value = reinterpret_cast<const char*>(sqlite3_column_text(stmt, 1));
00095         settings[name] = value;
00096     }
00097
00098     sqlite3_finalize(stmt);
00099     return settings;
00100 }
```

## 7.74 src/sine.cpp File Reference

```
#include <math.h>
#include "sine.h"
#include "tone.h"
#include "config.h"
```

Include dependency graph for sine.cpp:




## Macros

- #define M_PI (3.14159265)

### 7.74.1 Macro Definition Documentation

#### 7.74.1.1 M_PI

```
#define M_PI (3.14159265)
```

Definition at line 6 of file sine.cpp.

# 7.75 sine.cpp

Go to the documentation of this file.

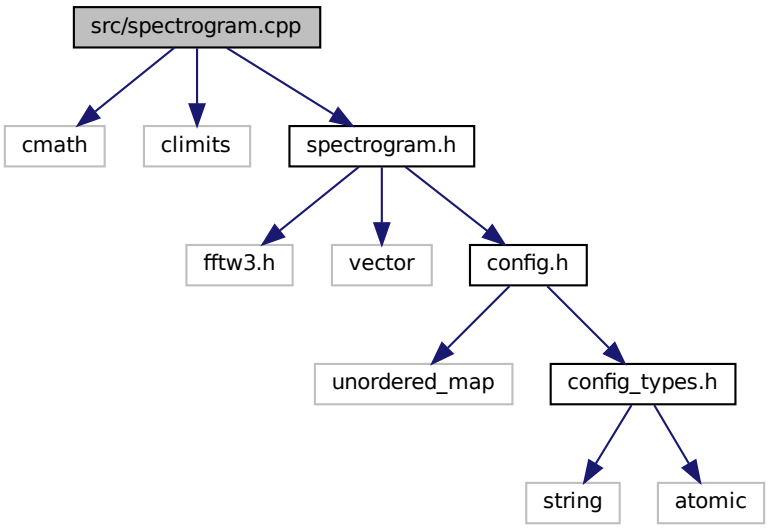
```
00001 #include <math.h>
00002 #include "sine.h"
00003 #include "tone.h"
00004
00005 #ifndef M_PI
00006 #define M_PI  (3.14159265)
00007 #endif
00008
00009 #include "config.h"
00010 Sine::Sine():audiocfg(Config::getInstance().audconf){
00011     rads_per_sample = 0.;
00012     prevfreq=0;
00013     phase=0.0;
00014 }
00015
00016 void Sine::setVisualizerUpdater(std::function<void(float)> updater){
00017     updateVisualizer = std::move(updater);
00018 }
00019
00020 void Sine::setMonoSignal(Tone& tone, float* monoBuffer[2]){
00021
00022     int frequency = tone.frequency.load();
00023     float amplitude = tone.volume.load();
00024
00025     if (frequency != prevfreq){ // reduce number of calculations
00026         rads_per_sample = (static_cast<float>(frequency * 2.* M_PI)) /
      static_cast<float>(audiocfg.sampleRate.load()); //radians traspotition per time unit
00027         prevfreq = frequency;
00028     }
00029
00030     for (int i=0;i<audiocfg.bufferSize.load();i++){
```

```
00031         float value = amplitude*(float)sin(phase);
00032         monoBuffer[0][i] = value;
00033         phase+=rads_per_sample;                 // shift phase by amount of rads_per_sample
00034         if (phase >= 2*M_PI) phase=0;           // if phase reaches 2pi , zero it down.
00035
00036         updateVisualizer(value); // fill the shareable ring buffer
00037     }
00038 }
00039
00040 void Sine::setStereoSignal(Tone& tone, float* stereoBuffer[2]){
00041
00042     int frequency = tone.frequency.load();
00043     float amplitude = tone.volume.load();
00044
00045     if (frequency != prevfreq){ // reduce number of calculations
00046         rads_per_sample = (static_cast<float>(frequency * 2.  * M_PI)) /
      static_cast<float>(audiocfg.sampleRate.load()); //radians traspotition per time unit
00047         prevfreq = frequency;
00048     }
00049
00050     for (int i=0;i<audiocfg.bufferSize.load();i++){
00051         float value = amplitude*(float)sin(phase);
00052         stereoBuffer[0][i] = value;
00053         stereoBuffer[1][i] = value;
00054         phase+=rads_per_sample;                 // shift phase by amount of rads_per_sample
00055         if (phase >= 2*M_PI) phase=0;           // if phase reaches 2pi , zero it down.
00056
00057         updateVisualizer(value); // fill the shareable ring buffer
00058     }
00059 }
```

## 7.76 src/spectrogram.cpp File Reference

```
#include <cmath>
#include <climits>
#include "spectrogram.h"
```

Include dependency graph for spectrogram.cpp:




## 7.77 spectrogram.cpp

Go to the documentation of this file.

```
00001 #include <cmath>
00002 #include <climits>
00003 #include "spectrogram.h"
```

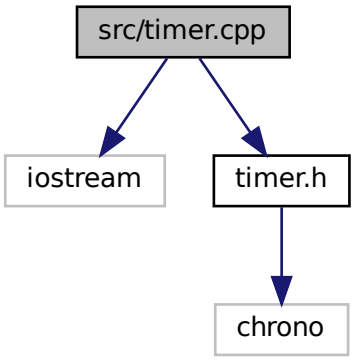
```
00004
00005 Spectrogram::Spectrogram():  readpos(0), writepos(0), fft_in(700){
00006
00007   calculateNFFT();
00008
00009   initialize_hamming(numAudioSamples);
00010
00011   auto fifoSize {numAudioSamples*2};
00012   ringBuffer.reserve(fifoSize);
00013   ringBuffer.resize(fifoSize);
00014
00015   // fft_in = static_cast<fftw_complex*> (fftw_malloc(sizeof(fftw_complex) * numAudioSamples));
00016   fft_in.reserve(numAudioSamples);
00017   fft_in.resize(numAudioSamples);
00018   fft_out = static_cast<fftw_complex*> (fftw_malloc(sizeof(fftw_complex) * numFFTPoints));
00019
00020   plan = fftw_plan_dft_r2c_1d(numAudioSamples, fft_in.data(), fft_out, FFTW_ESTIMATE); //FFTW_MEASURE
00021
00022   minMagnitude = INT_MAX, maxMagnitude = 0.;
00023
00024 }
00025
00026 void Spectrogram::calculateNFFT(){
00027   int W = cfg.dispconf.dispResW.load();
00028   int n = 2;
00029   // n^round(log_n(x)), where log_n(x) = log(x) / log(n)
00030   // numFFTPoints = std::pow(n, std::round( std::log(W) / std::log(n) )) ;
00031   numAudioSamples = static_cast<int>(std::pow(n, std::floor( std::log(W) / std::log(n) ))) ; // the
      closest power of two to a width (lower than width)
00032   numFFTPoints = numAudioSamples / 2 + 1;
00033 }
00034
00035 void Spectrogram::initialize_hamming(int n){
00036   hamming_window.reserve(n);
00037   hamming_window.resize(n);
00038   for (int i=0;i<n;++i)
00039     hamming_window[i]=0.54f-0.46f* static_cast<float>(cos(2*PI*i/(n-1)));
00040 }
00041
00042 Spectrogram::~Spectrogram(){
00043   if (plan) fftw_destroy_plan(plan);
00044   fftw_cleanup();
00045   // fftw_free(fft_in);
00046   fftw_free(fft_out);
00047 }
00048
00049 int Spectrogram::get_numAudioSamples(){
00050   return numAudioSamples;
00051 }
00052
00053 int Spectrogram::get_numFFTPoints(){
00054     return numFFTPoints;
00055 }
00056
00057 bool Spectrogram::write(const float& arg){ // T&& arg --> rvalue reference
00058     auto writePos = writepos.load(); //??
00059     auto nextWritePos = (writePos + 1) % ringBuffer.size();
00060
00061     // if the buffer is full, overwrite the oldest data
00062     if (nextWritePos == readpos.load()) {
00063         auto readPos = (readpos.load() + 1) % ringBuffer.size();
00064         readpos.store(readPos, std::memory_order_release);
00065     }
00066
00067     // write data to the buffer
00068     ringBuffer[writePos] = arg;
00069     writepos.store(nextWritePos);
00070     return true;
00071
00072 }
00073
00074 bool Spectrogram::readBatch(std::vector<float>& result, float &min_magnitude, float &max_magnitude) {
00075
00076     // Calculate the start index for reading the last N samples in a forward manner
00077     size_t startIndex = (writepos.load() - numAudioSamples + 1 + ringBuffer.size()) %
      ringBuffer.size();
00078
00079     // Read the N samples starting from startIndex
00080     size_t readPos = startIndex;
00081     for (size_t i = 0; i < (size_t)numAudioSamples; ++i) {
00082
00083         // fft_in[i][0]=ringBuffer[readPos] * hamming_window[i];
00084         // fft_in[i][1]=0;
00085         fft_in[i]=ringBuffer[readPos] * hamming_window[i];
00086
00087         readPos = (readPos + 1) % ringBuffer.size();  // Move to the next sample (circular)
00088     }
```

```
00089
00090     fftw_execute(plan);
00091
00092     for (int i = 0; i < numFFTPoints; ++i) {
00093         // Magnitude of the complex number
00094         float magnitude = static_cast<float>(std::sqrt(fft_out[i][0] * fft_out[i][0] + fft_out[i][1] *
      fft_out[i][1]));
00095
00096         minMagnitude = std::min(minMagnitude,magnitude);
00097         maxMagnitude = std::max(maxMagnitude,magnitude);
00098
00099         result[i] = magnitude;
00100     }
00101
00102     min_magnitude = minMagnitude;
00103     max_magnitude = maxMagnitude;
00104
00105     // Update readpos to point to the next sample to be read
00106     readpos.store((readPos + 1) % ringBuffer.size());
00107
00108     return true;
00109 }
```

## 7.78 src/timer.cpp File Reference

```
#include <iostream>
#include "timer.h"
```

Include dependency graph for timer.cpp:




## 7.79 timer.cpp

Go to the documentation of this file.

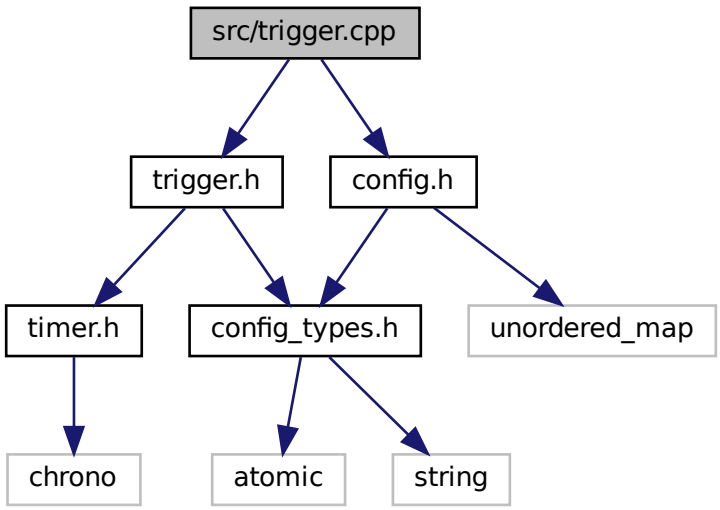
```
00001 #include <iostream>
00002 #include "timer.h"
00003
00004 Timer::Timer() :  initialSeconds(0), remainingTimeMilliseconds(0){}
00005
00006 void Timer::setTimer(int seconds) {
00007     initialSeconds = seconds;
00008     remainingTimeMilliseconds = seconds * 1000;
00009     lastUpdateTime = std::chrono::steady_clock::now();
00010 }
00011
00012 void Timer::start(){
00013     lastUpdateTime = std::chrono::steady_clock::now();
00014 }
00015
00016 bool Timer::update(int& secondsElapsed) {
00017
00018     // Calculate time passed since last update
```

```
00019     auto now = std::chrono::steady_clock::now();
00020     std::chrono::duration<int64_t, std::milli> elapsed =
      std::chrono::duration_cast<std::chrono::milliseconds>(now - lastUpdateTime);
00021
00022     // Subtract elapsed milliseconds from the remaining time
00023     remainingTimeMilliseconds -= static_cast<int>(elapsed.count());
00024     lastUpdateTime = now;  // Update last time frame
00025
00026     // If the timer has finished
00027     if (remainingTimeMilliseconds <= 0) {
00028         remainingTimeMilliseconds = 0;
00029         secondsElapsed = initialSeconds;  // Set elapsed time to the initial value
00030         return true;
00031     }
00032
00033     // Return the elapsed time in seconds (rounded)
00034     secondsElapsed = remainingTimeMilliseconds;
00035     return false;
00036 }
00037
00038 // Function to get the remaining time for display or logging purposes
00039 int Timer::getRemainingTime()const {
00040     return remainingTimeMilliseconds / 1000;  // Convert milliseconds back to seconds
00041 }
00042
00043 // Function to check if the timer has finished
00044 bool Timer::isTimerFinished()const {
00045     return remainingTimeMilliseconds <= 0;
00046 }
00047
```

## 7.80 src/trigger.cpp File Reference

```
#include "trigger.h"
#include "config.h"
```

Include dependency graph for trigger.cpp:




## 7.81 trigger.cpp

Go to the documentation of this file.

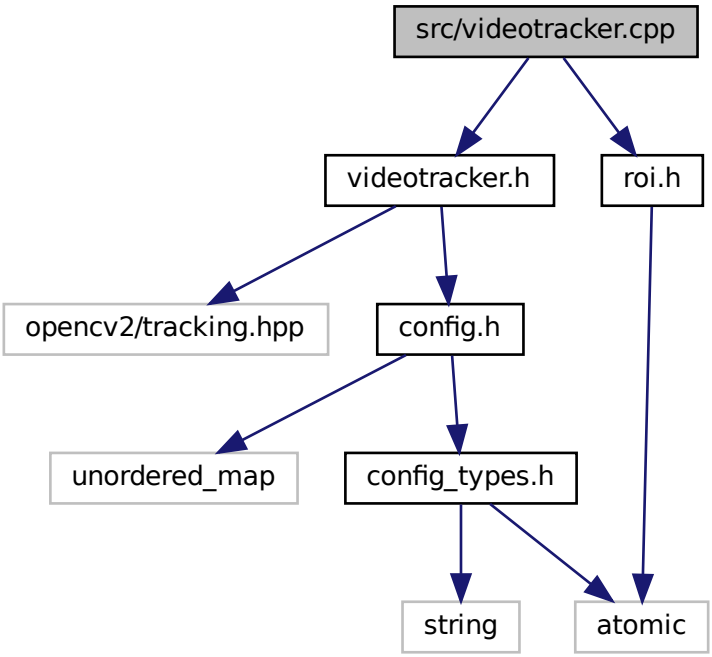
```
00001 #include "trigger.h"
00002 #include "config.h"
00003
```

```
00004 Trigger::Trigger() :  iavcfg
      (Config::getInstance().iavconf),timeDurationSec(photo_countdown_sec),mode(0)
00005 {
00006
00007     timer.setTimer(timeDurationSec);
00008
00009     // if (iavcfg.trigger == "Auto"){}
00010     // else if (iavcfg.trigger == "Manual"){}
00011 }
00012
00013 void Trigger::_modeToggle(){
00014     mode =!mode;
00015     timeDurationSec = (mode) ?  experience_duration_sec :  photo_countdown_sec;
00016 }
00017
00018 bool Trigger::isTrackingEnabled(float &remaining_percentage){
00019
00020         if (timer.isTimerFinished()){
00021             _modeToggle();
00022             timer.setTimer(timeDurationSec);
00023         }
00024
00025         int millisecondsElapsed;
00026
00027         timer.update(millisecondsElapsed);
00028         remaining_percentage = (static_cast<float> (millisecondsElapsed) /
      static_cast<float>(timeDurationSec*1000));
00029
00030         return mode;
00031 }
00032
00033 void Trigger::reset(){
00034     mode = 0;
00035     timeDurationSec = photo_countdown_sec;
00036 }
00037
00038 Timer* Trigger::getTimer(){
00039     return &timer;
00040 }
```

## 7.82 src/videotracker.cpp File Reference

```
#include "videotracker.h"
#include "roi.h"
```

Include dependency graph for videotracker.cpp:




## 7.83 videotracker.cpp

Go to the documentation of this file.

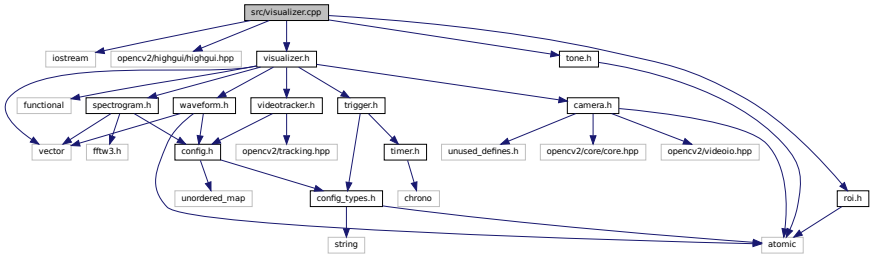
```
00001 #include "videotracker.h"
00002 #include "roi.h"
00003
00004 VideoTracker::VideoTracker(){
00005
00006     int radius=cfg.iavconf.roiRadius;
00007     int W=cfg.camconf.camResW.load();
00008     int H=cfg.camconf.camResH.load();
00009     cv::Rect temp((W/2)-radius,(H/2)-radius,radius*2,radius*2);
00010     centerBox=temp;
00011     boundingBox=temp;
00012 }
00013
00014
00015 void VideoTracker::initializeTracker(const cv::Mat& frame){
00016     if (cfg.iavconf.trackingAlg == "CSRT") {
00017         tracker = cv::TrackerCSRT::create();
00018     } else if (cfg.iavconf.trackingAlg == "KCF") {
00019         tracker = cv::TrackerKCF::create();
00020     }
00021     boundingBox = centerBox;
00022     tracker->init(frame, boundingBox);
00023 }
00024
00025 bool VideoTracker::trackObject(const cv::Mat &frame , RegionOfInterest &trackingSig){
00026
00027         bool trackingUpdated = tracker->update(frame, boundingBox);
00028         if (trackingUpdated){
00029
00030     // @ comment :  this gives the center x,y and the w and h
00031             trackingSig.centerX.store(static_cast<int>(boundingBox.x + boundingBox.width/2));
00032             trackingSig.centerY.store(static_cast<int>(boundingBox.y + boundingBox.height/2));
00033             trackingSig.volumeW.store(static_cast<int>(boundingBox.width));
00034             trackingSig.volumeH.store(static_cast<int>(boundingBox.height));
00035     // @ comment :  this gives the TOPLEFT corner x,y and the w and h
00036             // trackingSig.centerX.store(static_cast<int>(boundingBox.x));
```

```
00037             // trackingSig.centerY.store(static_cast<int>(boundingBox.y));
00038             // trackingSig.volumeW.store(static_cast<int>(boundingBox.width));
00039             // trackingSig.volumeH.store(static_cast<int>(boundingBox.height));
00040 
00041         }
00042         return trackingUpdated;
00043 
00044 }
```

## 7.84 src/visualizer.cpp File Reference

```
#include <iostream>
#include <opencv2/highgui/highgui.hpp>
#include "visualizer.h"
#include "roi.h"
#include "tone.h"
```

Include dependency graph for visualizer.cpp:




### Variables

- constexpr int qASCII {113}

### 7.84.1 Variable Documentation

#### 7.84.1.1 qASCII

```
constexpr int qASCII {113}  [constexpr]
```

Definition at line 8 of file visualizer.cpp.

## 7.85 visualizer.cpp

Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <opencv2/highgui/highgui.hpp>
00003 #include "visualizer.h"
00004 #include "roi.h"
00005 #include "tone.h"
00006 
00007 
00008 constexpr int qASCII {113}; // 113 q
00009 // constexpr int spaceASCII {32};// 32 space
00010 
00011 Visualizer::Visualizer(){
00012 
00013     int W=cfg.dispconf.dispResW.load();
00014     int H=cfg.dispconf.dispResH.load();
00015 
00016     cv::namedWindow("Interactive Audio Visualizer",cv::WINDOW_AUTOSIZE);
00017     cv::Mat img( H , W, CV_8UC3,cv::Scalar(0,0,0));
00018     visualFrame = img;
00019 
00020     _create_camMask();
```

```
00021
00022     int cameraW = cfg.camconf.camResW.load();
00023     int cameraH = cfg.camconf.camResH.load();
00024     LR = W - cameraW;
00025     TB = H - cameraH;
00026
00027     transpose_ratio_x = static_cast<float>(W) / static_cast<float>(cameraW) / 2.0f;
00028     transpose_ratio_y = static_cast<float>(H) / static_cast<float>(cameraH) / 2.0f;
00029
00030     trackingToggle = false;
00031
00032     int nfft = spectrogram.get_numFFTPoints();
00033     specMagnitude.reserve(nfft);
00034     specMagnitude.resize(nfft);
00035
00036     _set_freq_midBoundaries();
00037 }
00038
00039 void Visualizer::setAudiolizerUpdater(std::function<void(const bool, const bool, const
      RegionOfInterest&, Tone&)> function){
00040     updateAudioLizer = std::move(function);
00041 }
00042
00043 void Visualizer::updateTrackingMode(bool trackingEnabled){
00044
00045     if (trackingToggle!=trackingEnabled){
00046         if (!trackingToggle && trackingEnabled){
00047             videoTracker.initializeTracker(cameraFrame);
00048         }
00049         trackingToggle = trackingEnabled;
00050
00051     }
00052
00053 }
00054
00055 void Visualizer::updateAudioSignal(float newValue){
00056     waveform.write(newValue);
00057     spectrogram.write(newValue);
00058 }
00059
00060 void Visualizer::_create_camMask(){
00061
00062     int cameraW = cfg.camconf.camResW.load();
00063     int cameraH = cfg.camconf.camResH.load();
00064     int W=cfg.dispconf.dispResW.load();
00065     int H=cfg.dispconf.dispResH.load();
00066
00067     // calculate areas
00068     // int r = (cameraW>cameraH) ? cameraH/2 :  cameraW/2;
00069     int r = std::min(cameraW , cameraH)/2;
00070
00071     // int outArea = pow( 2*r, 2 ) - (M_PI * pow(r,2)); // pow( 2*r, 2 ) is the area of the box (same
      center, 2*r both edges) which is subtracted by the circle area r^2
00072     // outArea+= (abs(cameraW-cameraH) * r); // abs(cameraW-cameraH) = rest area outside the camera
      frame
00073
00074     int center_x = cameraW/2;
00075     int center_y = cameraH/2;
00076
00077     numPointsPerimeter = static_cast<int>(floor((sqrt(2)*(r-1)+4)/2)*8);  //
      https://stackoverflow.com/a/14995443/15842840
00078
00079     cv::Mat m1 = cv::Mat(cameraH,cameraW, CV_64F, cv::Scalar(0)); // CV_32F
00080     camBinaryMask=m1;
00081
00082     r = cfg.iavconf.roiRadius;
00083     int thickness = 1;
00084     circle( camBinaryMask,
00085         cv::Point(center_x,center_y),
00086         r,
00087         cv::Scalar( 0, 255, 0 ),
00088         thickness,
00089         cv::LINE_8);
00090
00091     for (int i=0;i<cameraW;i++){
00092         for (int j=0;j<cameraH;j++){
00093
00094             float center_dist = (float) sqrt ( pow((i-center_x),2) + pow((j-center_y),2) );
00095
00096             // https://stackoverflow.com/a/839931/15842840
00097             // Get the max distance for each point to normalize the distance between square and
      circles perimeter
00098             // x = cx + r * cos(a)
00099             // y = cy + r * sin(a)
00100             float max_dist = (float) sqrt ( pow((i-center_x),2) + pow((j-center_y),2) );
00101             bool condition = center_dist > (float)r ;
00102
```

```
00103             //(x - a)**2 + (y - b)**2 == r**2;
00104             // double term1 = (pow((i - cameraW/2),2) + pow((j - cameraH/2),2));
00105             // double term2 = pow(r,2);
00106             // bool condition2 = term1 >= term2;
00107             // bool condition2 = (pow((i - cameraW/2),2) + pow((j - cameraH/2),2)) >=
00108
00109             if (condition){
00110                 float transparency = (center_dist- static_cast<float>(r))/
    static_cast<float>(sqrt(pow(cameraW-cameraH,2)));
00111
00112                 camBinaryMask.at<double>(j,i) = transparency;
00113
00114                 // if (max_dist == r+4 || center_dist == r+3 || center_dist == r+1 || center_dist ==
    r+2 || center_dist == r){
00115                 if (max_dist == static_cast<float>(r+4) || center_dist == static_cast<float>(r+3) ||
00116                     center_dist == static_cast<float>(r+1) || center_dist == static_cast<float>(r+2)
    ||
00117                     center_dist == static_cast<float>(r)){
00118
00119                     int T = (H - cameraH)/2;
00120                     int L = (W - cameraW)/2;
00121                     int x = L + i;
00122                     int y = T + j;
00123                     visualFrame.at<cv::Vec3b>(y,x)[0] = 137;
00124                     visualFrame.at<cv::Vec3b>(y,x)[1] = 137;
00125                     visualFrame.at<cv::Vec3b>(y,x)[2] = 137;
00126                 }
00127             }
00128         }
00129     }
00130 }
00131
00132 void Visualizer::_set_freq_midBoundaries(){
00133     // set mid frequencies
00134     int minFreq  = cfg.iavconf.minFrequency;
00135     int maxFreq = cfg.iavconf.maxFrequency;
00136
00137     int frange = maxFreq - minFreq;
00138     leftMidFreq  = minFreq + (frange/3);
00139     rightMidFreq = minFreq + (2*frange/3);
00140 }
00141
00142 void Visualizer::broadcast(){
00143
00144     bool trackingEnabled,trackingUpdated;
00145     RegionOfInterest trackingSig;
00146     Tone tone;
00147
00148     while(true){
00149
00150         // camera in visualizer
00151         bool frameElapsed = camera.capture(cameraFrame); // get data
00152         if (!frameElapsed){
00153             return;
00154         }
00155
00156         float remaining_percentage;
00157         trackingEnabled = trigger.isTrackingEnabled(remaining_percentage);
00158         updateTrackingMode(trackingEnabled);
00159
00160         if (trackingEnabled){ // preprocess visual_frame -->   doesn't depict the frame, it just edits
    it so it does not require a new frame to be captured by the camera.
00161
00162             trackingUpdated = videoTracker.trackObject(cameraFrame, trackingSig);
00163
00164             if (!trackingUpdated){
00165                 // reset
00166                 trigger.reset();
00167             }
00168
00169             // update the current visualframe according to the changing of the tracking stimulus
00170             _set_BG_manually(tone);
00171             _set_FG_manually(trackingSig);
00172
00173
00174         }else{
00175             _setToCamera(remaining_percentage);
00176             trackingUpdated = trackingEnabled = false;
00177         }
00178
00179         updateAudioLizer(trackingUpdated, trackingEnabled, trackingSig, tone);
00180
00181         bool exit_msg = _showFrame();
00182         if (exit_msg)
00183             break;
00184     }
00185 }
```

```
00186
00187 Visualizer::~Visualizer(){
00188     cv::destroyWindow("Interactive Audio Visualizer");
00189     visualFrame.release();
00190     cameraFrame.release();
00191     camBinaryMask.release();
00192 }
00193
00194 bool Visualizer::_showFrame(){
00195     cv::imshow("Interactive Audio Visualizer", visualFrame);
00196     int msg = cv::waitKey(1);
00197     if (msg == qASCII) return true;
00198     // else if (cfg.iavconf.trigger=="Manual" && msg == spaceASCII)
00199     return false;
00200 }
00201
00202 void Visualizer::_set_BG_manually(Tone &tone){
00203
00204     int frequency = tone.frequency.load();
00205     float volume = tone.volume.load();
00206
00207     float percent;
00208
00209     B = G = R = 0;
00210     if (frequency> cfg.iavconf.minFrequency && frequency<=leftMidFreq){                // keep blue

00211         percent = static_cast<float>(frequency)/ static_cast<float>(leftMidFreq);  // high trans
00212         B = static_cast<int>(255.f * volume );
00213         G = static_cast<int>(255.f * percent);
00214         R = static_cast<int>(255.f * (1.-percent));
00215     }
00216     else if (frequency >leftMidFreq && frequency <= rightMidFreq){       // keep green
00217         percent = static_cast<float>(frequency)/ static_cast<float>(rightMidFreq); // low trans
00218         B = static_cast<int>(255.f * percent);
00219         G = static_cast<int>(255.f * volume);
00220         R = static_cast<int>(255.*(1.-percent));
00221     }
00222     else if (frequency > rightMidFreq && frequency <= cfg.iavconf.maxFrequency) {    // keep red
00223         percent = static_cast<float>(frequency)/static_cast<float>(cfg.iavconf.maxFrequency);

00224         B = static_cast<int>(255.*percent);
00225         G = static_cast<int>(255.*(1.-percent));
00226         R = static_cast<int>(255.f * volume);
00227     }
00228     visualFrame.setTo( cv::Scalar( B, G, R ) );
00229 }
00230
00231 void Visualizer::drawSmallcircle(const RegionOfInterest &roi){
00232
00233
00234     int W=cfg.dispconf.dispResW.load();
00235     int H=cfg.dispconf.dispResH.load();
00236
00237     int roicenterX = roi.centerX.load();
00238     int roicenterY = roi.centerY.load();
00239     int roiVolumeW = roi.volumeW.load();
00240     int roiVolumeH = roi.volumeH.load();
00241
00242     int center_x = LR/2 + roicenterX; //
00243     int center_y = TB/2 + roicenterY; // H/2;
00244
00245     if ( center_x < W/2 ) //-W*2/3)
00246         center_x -= static_cast<int>(transpose_ratio_x * (static_cast<float>(W) / 2.0f -
      static_cast<float>(center_x)));
00247     else if ( center_x > W/2) //*2/3)
00248         center_x += static_cast<int>(transpose_ratio_x * (static_cast<float>(center_x) -
      static_cast<float>(W) / 2.0f));
00249     if ( center_y < H/2 )
00250         center_y -= static_cast<int>(transpose_ratio_y * (static_cast<float>(H) / 2.0f -
      static_cast<float>(center_y)));
00251     else if ( center_y > H/2 )
00252         center_y += static_cast<int>(transpose_ratio_y * (static_cast<float>(center_y) -
      static_cast<float>(H) / 2.0f));
00253
00254     cv::Point center(W - center_x, center_y);//Declaring the center point - Reflect horizontally
00255     int radius = roiVolumeW > roiVolumeH ? roiVolumeW/2 :  roiVolumeH/2; //Declaring the radius
00256     cv::Scalar line_Color(0, 0, 0);//Color of the circle
00257     int thickness = 2;//thickens of the line
00258     circle(visualFrame, center,radius, line_Color, thickness);//Using circle()functi
00259
00260 }
00261
00262 void Visualizer::draWaveform(){
00263
00264     int W=cfg.dispconf.dispResW.load();
00265     int H=cfg.dispconf.dispResH.load();
00266     int cameraW = cfg.camconf.camResW.load();
```

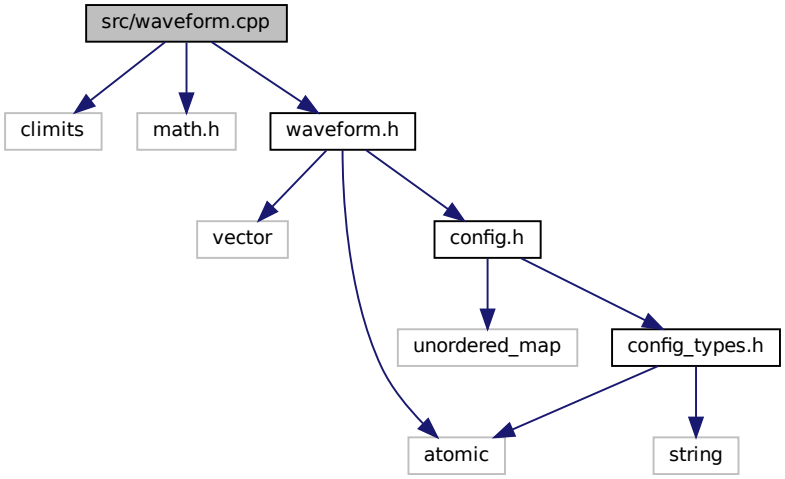
```
00267     int cameraH = cfg.camconf.camResH.load();
00268 
00269     int x_centre = W/2;
00270     int y_centre = H/2;
00271     int r = (cameraW>cameraH) ? cameraH/2 :  cameraW/2;
00272 
00273     float minMax[2];
00274     size_t numSamples;
00275     waveform.getMinMax(minMax);
00276     float min = minMax[0],max = minMax[1];
00277     numSamples = waveform.availableForReading();
00278 
00279 
00280     // depict waveform
00281     size_t end;
00282     double curRadians=0.0;
00283     double radianStep=2*M_PI / (double)numPointsPerimeter;
00284 
00285     if (numSamples< static_cast<size_t>(numPointsPerimeter)){
00286         end=numSamples;
00287     }else end = static_cast<size_t>(numPointsPerimeter);
00288 
00289     int thickness=1;
00290     int x1,x2,y1,y2;
00291     float percent;
00292     float new_radius;
00293     float waveVal;
00294     for (size_t i=0; i<end ; i++){
00295 
00296         // pixels are calculated given the following equations:
00297         // x = cx + r * cos(a)
00298         // y = cy + r * sin(a)
00299         x1 = static_cast<int>((float)r * std::cos(curRadians) + (float)x_centre);
00300         y1 = static_cast<int>((float)r * std::sin(curRadians) + (float)y_centre);
00301 
00302         waveform.read(waveVal);
00303         // normalize in range [-1 1]
00304         percent = 2* ( (waveVal-min) / (max-min) ) -1;
00305 
00306         // trasport x and y
00307         new_radius = static_cast<float>(r + ((float)H / 40.  * percent));
00308 
00309         x2 = static_cast<int>(new_radius * std::cos(curRadians) + (float)x_centre);
00310         y2 = static_cast<int>(new_radius * std::sin(curRadians) + (float)y_centre);
00311 
00312         cv::Point p1(x1, y1), p2(x2, y2);
00313         cv::line(visualFrame, p1, p2, cv::Scalar(255, 0, 0), thickness, cv::LINE_8);
00314 
00315         curRadians+=radianStep;
00316     }
00317 
00318 }
00319 
00320 void Visualizer::_set_FG_manually(const RegionOfInterest &roi){
00321 
00322     drawSmallcircle(roi);
00323     draWaveform();
00324     drawSpectrogram();
00325 
00326 }
00327 
00328 void Visualizer::_show_timer(float percent) {
00329 
00330     int x = cameraFrame.cols / 2;
00331     int y = cameraFrame.rows / 2;
00332 
00333     int radius = cfg.iavconf.roiRadius;
00334     int thickness = 3;
00335     float angle = percent * 360.0f;
00336     cv::circle(cameraFrame, cv::Point(x, y), radius, CV_RGB(245, 245, 245), thickness);
00337 
00338     // int r = 255 - (millisecondsElapsed * 5) % 255;  // Gradually change the color to orange/yellow
00339     // int g = (millisecondsElapsed * 2) % 255;
00340     // int b = (millisecondsElapsed * 4) % 255;
00341     int r = static_cast<int>(127.5 * (1 + sin(angle * M_PI / 180.0)));  // Sinusoidal for red
00342     int g = static_cast<int>(127.5 * (1 + sin((angle + 120) * M_PI / 180.0)));  // Sinusoidal for
      green
00343     int b = static_cast<int>(127.5 * (1 + sin((angle + 240) * M_PI / 180.0)));  // Sinusoidal for blue
00344 
00345     cv::ellipse(cameraFrame, cv::Point(x, y), cv::Size(radius, radius), 0, -90, -90 + angle, CV_RGB(r,
      g, b), thickness);
00346 
00347 }
00348 
00349 void Visualizer::_setToCamera(float remaining_percentage){
00350 
00351     _show_timer(remaining_percentage);
```

```
00352
00353     int cameraW=cameraFrame.cols;
00354     int cameraH=cameraFrame.rows;
00355     // top left of the visualFrame is ...
00356     int L = (visualFrame.cols - cameraW)/2;
00357     int T = (visualFrame.rows - cameraH)/2;
00358
00359     // draw transparent pixels in a form of enclosed circle within camera frame
00360     double vB = (double)visualFrame.at<cv::Vec3b>(0,0)[0];
00361     double vG = (double)visualFrame.at<cv::Vec3b>(0,0)[1];
00362     double vR = (double)visualFrame.at<cv::Vec3b>(0,0)[2];
00363     for (int i=0;i<cameraW;i++){
00364         for (int j=0;j<cameraH;j++){
00365             if (camBinaryMask.at<double>(j,i)>0.){
00366                 double ratio = camBinaryMask.at<double>(j,i);
00367                 cameraFrame.at<cv::Vec3b>(j,i)[0] = static_cast<unsigned char>(((ratio*vB) +
      (1.-ratio)*(double)cameraFrame.at<cv::Vec3b>(j,i)[0])/2.);
00368                 cameraFrame.at<cv::Vec3b>(j,i)[1] = static_cast<unsigned char>(((ratio*vG) +
      (1.-ratio)*(double)cameraFrame.at<cv::Vec3b>(j,i)[1])/2.);
00369                 cameraFrame.at<cv::Vec3b>(j,i)[2] = static_cast<unsigned char>(((ratio*vR) +
      (1.-ratio)*(double)cameraFrame.at<cv::Vec3b>(j,i)[2])/2.);
00370             }
00371         }
00372     }
00373     cameraFrame.copyTo(visualFrame(cv::Rect(L,T,cameraW,cameraH)));
00374 }
00375
00376 void Visualizer::drawSpectrogram(){
00377
00378     const double FREQ_MIN = (double)cfg.iavconf.minFrequency;
00379     const double FREQ_MAX = (double)cfg.iavconf.maxFrequency;
00380     int SR = cfg.audconf.sampleRate.load();
00381     int W = cfg.dispconf.dispResW.load();
00382     int H = cfg.dispconf.dispResH.load();
00383     int numAudioSamples = spectrogram.get_numAudioSamples();
00384
00385     float min_magnitude,max_magnitude;
00386     spectrogram.readBatch(specMagnitude,min_magnitude,max_magnitude);
00387
00388     for (size_t i = 0; i < specMagnitude.size() ; ++i) {
00389
00390         float magnitude = specMagnitude[i];
00391
00392         // Minmax magnitude's normalization
00393         int normalized_magnitude = static_cast<int>(
00394                                         std::min(
00395                                             (float)((magnitude - min_magnitude) / (max_magnitude -
      min_magnitude) * static_cast<float>(H) * 0.5),
00396                                             static_cast<float>(H)/2.f
00397                                         )
00398                                     );
00399
00400         // Calculate the frequency interval for each bin
00401         double freq_bin = (SR / 2.0) * (static_cast<double>(i) / static_cast<double>(numAudioSamples /
      2.0)); // Frequency of the current bin
00402
00403         // filtering the frequencies outside the scope of the iav application
00404         if (freq_bin >= FREQ_MIN && freq_bin <= FREQ_MAX) {
00405
00406             int x = static_cast<int>((freq_bin - FREQ_MIN) / (FREQ_MAX - FREQ_MIN) * W);
00407
00408             // Create a the drawing line
00409             int line_length = normalized_magnitude;
00410             int top = H / 2 - line_length;
00411             int bottom = H / 2 + line_length;
00412
00413             for (int y = top; y <= bottom; ++y) {
00414                 // Ensure not out-of-bounds
00415                 if (y >= 0 && y < H && x >= 0 && x < W) {
00416                     visualFrame.at<cv::Vec3b>(y,x)[0] = 255 - B;
00417                     visualFrame.at<cv::Vec3b>(y,x)[1] = 255 - G;
00418                     visualFrame.at<cv::Vec3b>(y,x)[2] = 255 - R;
00419
00420                 }
00421             }
00422
00423         }
00424     }
00425
00426 }
00427
```

## 7.86 src/waveform.cpp File Reference

```
#include <climits>
#include <math.h>
#include "waveform.h"
```

Include dependency graph for waveform.cpp:




## 7.87 waveform.cpp

Go to the documentation of this file.

```
00001 #include <climits>
00002 #include <math.h>
00003 #include "waveform.h"
00004
00005 Waveform::Waveform():  readpos(0), writepos(0){
00006     // calculate num of buffers per display frame;
00007     int audioSamplesPerFrame = static_cast<int>(cfg.audconf.sampleRate.load() /
      cfg.dispconf.fps.load());
00008     int buffersPerFrame = std::ceil( static_cast<float>(audioSamplesPerFrame) /
      static_cast<float>(cfg.audconf.bufferSize.load()));
00009     audioSamplesPerFrame = (buffersPerFrame+1) * cfg.audconf.bufferSize.load(); // add 1 and make it
      divisible by bufferSize.
00010
00011     waveTable.reserve(audioSamplesPerFrame);
00012     waveTable.resize(audioSamplesPerFrame);
00013
00014     capacity = (size_t)audioSamplesPerFrame;
00015     min = INT_MAX, max = 0.;
00016 }
00017
00018 bool Waveform::write(const float& arg){ // T&& arg --> rvalue reference
00019     auto writePos = writepos.load(); //??
00020     auto nextWritePos = (writePos + 1) % waveTable.size();
00021
00022     // if the buffer is full, overwrite the oldest data
00023     if (nextWritePos == readpos.load()) {
00024         auto readPos = (readpos.load() + 1) % waveTable.size();
00025         readpos.store(readPos, std::memory_order_release);
00026     }
00027
00028     min = std::min(min,arg);
00029     max = std::max(max,arg);
00030
00031     // write data to the buffer
00032     waveTable[writePos] = arg;
00033     writepos.store(nextWritePos);
00034     return true;
00035
00036 }
```

```
00037
00038 bool Waveform::read(float& result){
00039
00040     auto readPos = readpos.load();
00041
00042     // if buffer is empty return false
00043     if (readPos == writepos.load()){
00044         return false;
00045     }
00046
00047     // move data out of the buffer;
00048     result = waveTable[readPos];
00049
00050     // advance the read pointer
00051     auto nextReadPos = (readPos + 1) % waveTable.size();
00052     readpos.store(nextReadPos);
00053     return true;
00054 }
00055
00056 bool Waveform::isEmpty()const {
00057     return readpos.load() == writepos.load();
00058 }
00059
00060 bool Waveform::isFull()const {
00061     return (( writepos.load() + 1 ) % waveTable.size()) == readpos.load();
00062 }
00063
00064 size_t Waveform::size()const {
00065     return capacity;
00066 }
00067
00068 size_t Waveform::availableForReading()const {
00069     size_t writePos = writepos.load();
00070     size_t readPos = readpos.load();
00071     if (writePos >= readPos) {
00072         return writePos - readPos;  // Normal case, no wraparound
00073     } else {
00074         return waveTable.size() - (readPos - writePos);  // Wraparound case
00075     }
00076 }
00077
00083 void Waveform::getMinMax(float minMax[2]){
00084     minMax[0] = this->min;
00085     minMax[1] = this->max;
00086 }
```