

pi-loop
0.9(prototype)

Generated by Doxygen 1.8.17

1 Main Page	1
2 Class Index	5
2.1 Class List	5
3 Class Documentation	7
3.1 AudioServer Class Reference	7
3.1.1 Detailed Description	7
3.1.2 Member Function Documentation	7
3.1.2.1 start()	7
3.1.2.2 stop()	7
3.2 Buttons Class Reference	8
3.2.1 Detailed Description	8
3.2.2 Member Function Documentation	8
3.2.2.1 is_pressed()	8
3.2.2.2 setup()	8
3.3 Channel Class Reference	8
3.3.1 Detailed Description	9
3.3.2 Member Function Documentation	9
3.3.2.1 add_track()	9
3.3.2.2 clean()	10
3.3.2.3 dub()	10
3.3.2.4 get_name()	10
3.3.2.5 get_num_tracks()	10
3.3.2.6 get_out_signal()	10
3.3.2.7 get_volume()	10
3.3.2.8 isEmpty()	11
3.3.2.9 rec()	11
3.3.2.10 reset()	11
3.3.2.11 reset_num_tracks()	11
3.3.2.12 set_name()	11
3.3.2.13 set_volume()	11
3.3.2.14 undub()	12
3.3.2.15 update_rec_signal()	12
3.3.2.16 volume_down()	12
3.3.2.17 volume_up()	12
3.4 Config Class Reference	12
3.4.1 Detailed Description	13
3.4.2 Member Function Documentation	13
3.4.2.1 display()	13
3.4.2.2 get_button_state()	13
3.4.2.3 get_curr_session()	14
3.4.2.4 get_max_sessions()	14

3.4.2.5 open()	14
3.4.2.6 reset()	14
3.4.2.7 save()	14
3.5 Effects Class Reference	15
3.5.1 Detailed Description	15
3.5.2 Member Function Documentation	15
3.5.2.1 apply()	15
3.5.2.2 initialize_effects()	15
3.5.2.3 toggle_effect()	16
3.6 Handshake Class Reference	16
3.6.1 Detailed Description	17
3.6.2 Member Function Documentation	17
3.6.2.1 activate()	17
3.6.2.2 check_status()	17
3.6.2.3 connect_input_device()	17
3.6.2.4 connect_output_device()	18
3.6.2.5 disconnect_client()	18
3.6.2.6 disconnect_input_device()	18
3.6.2.7 disconnect_output_device()	18
3.6.2.8 get_input_buffer() [1/2]	19
3.6.2.9 get_input_buffer() [2/2]	19
3.6.2.10 get_output_buffer() [1/2]	19
3.6.2.11 get_output_buffer() [2/2]	20
3.6.2.12 link_client()	20
3.6.2.13 prevent_failure()	20
3.6.2.14 register_devices()	20
3.6.2.15 register_input_port()	20
3.6.2.16 register_output_port()	21
3.6.2.17 set_buffer_size()	21
3.6.2.18 set_process_callback()	21
3.7 hardwareInterface Class Reference	21
3.7.1 Detailed Description	22
3.7.2 Member Function Documentation	22
3.7.2.1 deafen()	22
3.7.2.2 display()	22
3.7.2.3 get_display_initializer()	22
3.7.2.4 get_metro_display()	23
3.7.2.5 listen()	23
3.7.2.6 update()	23
3.8 Keyboard Class Reference	23
3.8.1 Detailed Description	24
3.8.2 Member Function Documentation	24

3.8.2.1 is_pressed()	24
3.9 Leds Class Reference	24
3.9.1 Detailed Description	25
3.9.2 Member Function Documentation	25
3.9.2.1 display()	25
3.9.2.2 initialize_leds()	25
3.9.2.3 perform_operation()	25
3.9.2.4 set_metro_state()	25
3.9.2.5 tick_tock()	26
3.9.2.6 turnOff()	26
3.10 Looper Class Reference	26
3.10.1 Detailed Description	27
3.10.2 Member Function Documentation	27
3.10.2.1 display_states()	27
3.10.2.2 get_metronome_state()	27
3.10.2.3 recdub()	27
3.10.2.4 reset()	28
3.10.2.5 save()	28
3.10.2.6 start_stop_all()	28
3.10.2.7 stoperase()	28
3.10.2.8 tap_alter_metronome()	29
3.10.2.9 update_buffer()	29
3.10.2.10 volume_change()	29
3.10.2.11 volume_down()	30
3.10.2.12 volume_up()	30
3.11 LooperState Struct Reference	30
3.11.1 Detailed Description	31
3.12 Menu Class Reference	31
3.12.1 Detailed Description	31
3.12.2 Member Function Documentation	31
3.12.2.1 load_session()	31
3.12.2.2 notify_menu()	32
3.12.2.3 set_display_initializer()	32
3.12.2.4 set_metro_display()	32
3.12.2.5 setup()	32
3.12.2.6 unload()	33
3.13 Metronome Class Reference	33
3.13.1 Detailed Description	33
3.13.2 Member Function Documentation	33
3.13.2.1 alter_tempo()	34
3.13.2.2 clear()	34
3.13.2.3 lock()	34

3.13.2.4 pause()	34
3.13.2.5 start_timing()	34
3.13.2.6 stop_timing()	34
3.13.2.7 sync()	35
3.13.2.8 tap_tempo()	35
3.13.2.9 tick_tock()	35
3.13.2.10 unlock()	35
3.13.2.11 unpause()	35
3.14 Mixer Class Reference	36
3.14.1 Detailed Description	36
3.14.2 Member Function Documentation	36
3.14.2.1 save_jam()	36
3.14.2.2 update_buffer()	36
3.15 Monitor Class Reference	37
3.15.1 Detailed Description	37
3.15.2 Member Function Documentation	37
3.15.2.1 connect()	37
3.15.2.2 disconnect()	38
3.15.2.3 get_states()	38
3.15.2.4 initialize_effects()	38
3.15.2.5 mute_input()	38
3.15.2.6 mute_output()	39
3.15.2.7 process()	39
3.15.2.8 set_stream_buffer()	39
3.15.2.9 toggle_effect()	39
3.15.2.10 toggle_states()	40
3.15.2.11 update_states()	40
3.16 PiLoop Class Reference	40
3.16.1 Detailed Description	41
3.16.2 Member Function Documentation	41
3.16.2.1 start_console()	41
3.17 Potentiometers Class Reference	41
3.17.1 Detailed Description	41
3.17.2 Member Function Documentation	41
3.17.2.1 is_changed()	41
3.17.2.2 setup()	42
3.18 Response Struct Reference	42
3.18.1 Detailed Description	42
3.18.2 Member Function Documentation	42
3.18.2.1 isEmpty()	43
3.18.2.2 reset()	43
3.19 Screen Class Reference	43

3.19.1 Detailed Description	43
3.19.2 Member Function Documentation	43
3.19.2.1 display()	43
3.19.2.2 initialize_states()	44
3.19.2.3 perform_operation()	44
3.19.2.4 tick_tock()	44
3.19.2.5 turnOff()	44
3.20 Session Class Reference	44
3.20.1 Detailed Description	45
3.20.2 Member Function Documentation	45
3.20.2.1 evacuate()	45
3.20.2.2 get_name()	45
3.20.2.3 load()	46
3.20.2.4 migrate()	46
3.20.2.5 notify_session()	46
3.20.2.6 save()	46
3.20.2.7 save_jam()	46
3.20.2.8 set_disp_initializer()	46
3.20.2.9 set_metronome_display()	47
3.20.2.10 set_name()	47
3.20.2.11 setup()	47
3.21 time_signature Struct Reference	47
3.21.1 Detailed Description	48
3.22 Trigger Struct Reference	48
3.22.1 Detailed Description	48
3.22.2 Member Function Documentation	48
3.22.2.1 reset()	48
3.23 UI Class Reference	49
3.23.1 Detailed Description	49
3.23.2 Member Function Documentation	49
3.23.2.1 initialize_display() [1/2]	49
3.23.2.2 initialize_display() [2/2]	50
3.23.2.3 listen_user() [1/2]	50
3.23.2.4 listen_user() [2/2]	50
3.23.2.5 metro_display() [1/2]	50
3.23.2.6 metro_display() [2/2]	51
3.23.2.7 show() [1/2]	51
3.23.2.8 show() [2/2]	51
3.23.2.9 turnOff() [1/2]	51
3.23.2.10 turnOff() [2/2]	52
3.23.2.11 update_output_state() [1/2]	52
3.23.2.12 update_output_state() [2/2]	52

Chapter 1

Main Page

A linux based audio looper written in C++.

[Description](#) | [Features](#) | [Documentation](#) | [Installation](#) | [Usage](#) | [Future_Work](#) | [Feedback](#) | [Related](#)

Description

Pi-Loop is a real-time audio looper application that enables users to create interactive musical sessions. One may use an external audio interface to connect a microphone and an instrument or to use the integrated sound card to test it.

This repo consists of 2 different **modes** that make use of alternative user interfaces to interact with the software:

- **PC mode**

This version runs directly on a Linux machine and allows the usage of **computer components** such as the **keyboard** **to trigger events**** and the **screen** **to access the output state of the program****.

- ****RPI mode**

This version runs on a ****Raspberry Pi**** and uses **GPIO**-based connected ****buttons** and **rotaries** **to enable user interaction with the program****, and ****LEDs** **to access the output state of the program****.

Both modes lie on the same codebase and can be configured on build. More information can be found in the [installation section](#) below.

Features

Key features that make [PiLoop](#) unique:

- **Input signal handling flexibility**

Individual ARM and [Monitor](#) (IN/OUT) states for each input channel allow users to choose on the fly which input channels will be streamed on the speakers and which will contribute to the creation of the current loop.

- **Jam saving feature**

Once decided, users can save their overall musical session as a wave file, by pressing a single button. Therefore, not only the loop stream is saved but the combined audio signal stream that has been streamed on the speakers during the session. This can be quite handy for musicians who want to save a session and use it to keep track of their inspirational dynamics or alternatively to use the saved file as a songwriting guide that can be further manipulated in a DAW afterward.

- **Individual effects on each input channel**

This increases the flexibility to add effects individually on each input channel.

Common features supported:

- 2 input channels*
- 1 stereo output channel*
- 3 fully operational loop channels (read further in the manual page)
- metronome
 - tap tempo
 - alter tempo
- a menu feature to load, change, and save session presets.

****depends on the number of IO channels supported by the audio interface used.***

Documentation

Documentation is uploaded as a [pdf](#) form.

To use an `html` version, install and run `doxygen` to generate the documentation files.

```
cd pi-loop-console
doxygen Doxyfile
```

The documentation files will be stored within the `files/docs/html` directory. Open the `files/docs/html/index.↵.html` in a browser window to overview it.

If you are also interested in the code design, you may want to check the [UML file](#).

Installation

Follow this link to access information on *installing*, *configuring*, and *building* [PiLoop](#).

Additionally, to setup Pi-Loop in **RPI mode** on a Raspberry Pi, follow this link to obtain an overview on the connections made to the GPIO pins.

Usage

To run [PiLoop](#), you have to run with sudo privilege:

```
$ sudo ./piloop
```

To see how to use it, visit the manual page

Future_Work

The future work listing is summarized as follows:

- Fix found bugs.
- Improve existing features
- Add new features
- Upgrade

Visit the future-work list to get informed about what's next.

Feedback

If you have used this application and would like to contribute or share your feedback and recommendations on improvements, or if there is anything you would like to say about it by any means, please don't hesitate to share it on [Gitter](#) or contact me at melissaspaschalis@gmail.com.

Related

If you liked this project, you may also like:

- [Raspberry Pi Looper synth drum thing](#)
- [making a looper with pure data](#)
- [an interactive audio visualizer](#)

```
:drum: :desktop_computer: :guitar: :trumpet: :banjo: :violin: :musical_keyboard: :accordion: ↔
:saxophone: :headphones: :musical_note: :notes: :smiley: Enjoy PiLoop :smiley: :notes: :musical_note:
:headphones: :saxophone: :accordion: :musical_keyboard: :violin: :banjo: :trumpet: :guitar: :desktop_↔
computer: :drum:
```

Chapter 2

Class Index

2.1 Class List

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

AudioServer	The jack-audio server running on the alsa drivers	7
Buttons	Buttons as the GPIO-based input interface	8
Channel	Defines a looper-channel designed to loop tracks	8
Config	The config module responsible for saving, loading, and initializing session data	12
Effects	The effects module. Currently unsupported	15
Handshake	The jack-audio client	16
hardwareInterface	The generic interface of the pi-loop. It has an input and an output interface	21
Keyboard	The keyboard as the computer-based input interface	23
Leds	LEDs as the GPIO-based output interface	24
Looper	Constisted of 3 loop-channels	26
LooperState	Stores the looper state	30
Menu	The programs menu that acts as the session manager	31
Metronome	A metronome defined within Looper	33
Mixer	The mixer module responsible to stream the output signal	36
Monitor	The monitor module encapsulates the jack audio client	37
PiLoop	The threading manager. It starts the program	40
Potentiometers	Potentiometers to adjust the looper-channel's volume	41
Response	Defines a response message destined to the displayer module of the program	42
Screen	The screen as the computer-based output interface	43
Session	Defines a musical session. It is consisted of a Monitor , a Looper and a Mixer	44

time_signature	Struct to hold the information of rythm	47
Trigger	Holds the triggers of the user	48
UI	Defines a computer based interface constisted of a keyboard and the screen	49

Chapter 3

Class Documentation

3.1 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.
- void [start](#) ()
Starts the jack audio server.
- void [stop](#) ()
Stops the jack audio server.

3.1.1 Detailed Description

The jack-audio server running on the alsa drivers.

3.1.2 Member Function Documentation

3.1.2.1 start()

```
void AudioServer::start ( )
```

Starts the jack audio server.

Returns

void

3.1.2.2 stop()

```
void AudioServer::stop ( )
```

Stops the jack audio server.

Returns

void

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

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

3.2 Buttons Class Reference

[Buttons](#) as the GPIO-based input interface.

```
#include <buttons.h>
```

Public Member Functions

- [Buttons](#) ()
Class constructor.
- void [setup](#) (int)
Method used as an implicit constructor to setup a buttons instance.
- void [is_pressed](#) (Trigger &)
Function to return if a button is pressed by the user.

3.2.1 Detailed Description

[Buttons](#) as the GPIO-based input interface.

3.2.2 Member Function Documentation

3.2.2.1 is_pressed()

```
void Buttons::is_pressed (
    Trigger & trigger )
```

Function to return if a button is pressed by the user.

Parameters

<i>Trigger&</i>	- the trigger to be returned.
---------------------	-------------------------------

Returns

void

3.2.2.2 setup()

```
void Buttons::setup (
    int base )
```

Method used as an implicit constructor to setup a buttons instance.

Parameters

<i>int</i>	- the base number for the mcp23017 IO expander
------------	--

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

- include/gpio/buttons.h
- src/gpio/buttons.cpp

3.3 Channel Class Reference

Defines a looper-channel designed to loop tracks.

```
#include <channel.h>
```


Public Member Functions

- [Channel](#) ()
default constructor
- void [reset](#) ()
function used to clear all data written on the channels.
- void [set_name](#) (const char *)
function used to set a name on the current channel.
- const char * [get_name](#) ()
function used to get the name of the current channel.
- void [add_track](#) ()
function used to add a new track on the current channel. In fact, it only stores the information that the track is saved.
- void [reset_num_tracks](#) ()
sets num tracks to zero.
- int [get_num_tracks](#) ()
gets the number of tracks of this channel.
- void [update_rec_signal](#) (float[BUFFER_SIZE], int)
function to pass the current buffer to rec or to dub. This function is only called when there is work to be done.
- float [get_volume](#) ()
gets the volume of the current channel
- void [set_volume](#) (int)
sets the volume for the current channel
- void [volume_up](#) ()
Used in PC mode to increase the volume of channel by <volumeStep> percent.
- void [volume_down](#) ()
Used in PC mode to decrease the volume of channel by <volumeStep> percent.
- void [clean](#) ()
function used to clean recorded data
- void [rec](#) (float[BUFFER_SIZE], int)
function to record data. It is used when channel has no tracks on it yet.
- void [dub](#) (float[BUFFER_SIZE], int)
function to store new data on top of previously recorded data.
- void [undub](#) ()
function used to erase last channel's record.
- std::vector< float > [get_out_signal](#) (int)
function to return the channels output signal
- bool [isEmpty](#) ()
function to return whether the channel has recorded tracks or not

3.3.1 Detailed Description

Defines a looper-channel designed to loop tracks.

3.3.2 Member Function Documentation

3.3.2.1 [add_track\(\)](#)

```
void Channel::add_track ( )
```

function used to add a new track on the current channel. In fact, it only stores the information that the track is saved.

Returns

void

3.3.2.2 clean()

```
void Channel::clean ( )
```

function used to clean recorded data

Returns

void

3.3.2.3 dub()

```
void Channel::dub (
    float monitor[BUFFER_SIZE],
    int idx )
```

function to store new data on top of previously recorded data.

Returns

void

3.3.2.4 get_name()

```
const char * Channel::get_name ( )
```

function used to get the name of the current channel.

Returns

void

3.3.2.5 get_num_tracks()

```
int Channel::get_num_tracks ( )
```

gets the number of tracks of this channel.

Returns

int - the number of tracks

3.3.2.6 get_out_signal()

```
std::vector< float > Channel::get_out_signal (
    int idx )
```

function to return the channels output signal

Parameters

<i>int</i>	- the index that indicates the current playback position
------------	--

Returns

std::vector<float> - the output buffer

3.3.2.7 get_volume()

```
float Channel::get_volume ( )
```

gets the volume of the current channel

Returns

float - the volume as type of a float percentage

3.3.2.8 isEmpty()

```
bool Channel::isEmpty ( )
```

function to return whether the channel has recorded tracks or not

Returns

bool - true if has tracks, otherwise false.

3.3.2.9 rec()

```
void Channel::rec (
    float monitor[BUFFER_SIZE],
    int idx )
```

function to record data. It is used when channel has no tracks on it yet.

Returns

void

3.3.2.10 reset()

```
void Channel::reset ( )
```

function used to clear all data written on the channels.

Returns

void

3.3.2.11 reset_num_tracks()

```
void Channel::reset_num_tracks ( )
```

sets num tracks to zero.

Returns

void

3.3.2.12 set_name()

```
void Channel::set_name (
    const char * name )
```

function used to set a name on the current channel.

Returns

void

3.3.2.13 set_volume()

```
void Channel::set_volume (
    int volume )
```

sets the volume for the current channel

Parameters

<i>int</i>	- the volume as type of an int percentage
------------	---

Returns

void

3.3.2.14 undub()

```
void Channel::undub ( )
```

function used to erase last channel's record.

Returns

void

3.3.2.15 update_rec_signal()

```
void Channel::update_rec_signal (
    float monitor[BUFFER_SIZE],
    int idx )
```

function to pass the current buffer to rec or to dub. This function is only called when there is work to be done.

Returns

void

3.3.2.16 volume_down()

```
void Channel::volume_down ( )
```

Used in PC mode to decrease the volume of channel by <volumeStep> percent.

Returns

void

3.3.2.17 volume_up()

```
void Channel::volume_up ( )
```

Used in PC mode to increase the volume of channel by <volumeStep> percent.

Returns

void

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

- include/channel.h
- src/channel.cpp

3.4 Config Class Reference

The config module responsible for saving, loading, and initializing session data.

```
#include <config.h>
```

Public Member Functions

- void `open` (int)
Reads the information stored within the configuration file for a single session.
- void `save` ()
Saves the information of the current session within the configuration file.
- void `display` ()
Displays the information of the current session.
- void `reset` ()
Discards all the changes made during a session.
- bool `get_button_state` (const Control &)
getter for button states. Used only for initialization.
- void `toggle_button_state` (const Control &)
Toggles the button states. Discarded since not thread safe.
- int `get_max_sessions` ()
Method to read the number of session defined within the configuration file.
- int `get_curr_session` ()
Method to return the id of the current session.

Static Public Member Functions

- static Config & `getInstance` ()
Method to return a reference to the Singleton object.

Public Attributes

- std::string `currSession_name`
- std::unordered_map< Control, bool > `button_states`
- float `tempo`
- int `rythm_numerator`
- int `rythm_denominator`
- std::string `jam_savepath`
- std::string `session_savepath`

3.4.1 Detailed Description

The config module responsible for saving, loading, and initializing session data.

3.4.2 Member Function Documentation

3.4.2.1 `display()`

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

Displays the information of the current session.

Returns

```
void
```

3.4.2.2 `get_button_state()`

```
bool Config::get_button_state (
    const Control & ctl )
```

getter for button states. Used only for initialization.

Parameters

<i>const</i>	Control& - the related button of interest
--------------	---

Returns

bool - the state of the current button

3.4.2.3 get_curr_session()

```
int Config::get_curr_session ( )
```

Method to return the id of the current session.

Returns

int - the id of the current session.

3.4.2.4 get_max_sessions()

```
int Config::get_max_sessions ( )
```

Method to read the number of session defined within the configuration file.

Returns

int - the maximum number of sessions.

3.4.2.5 open()

```
void Config::open (
    int curr_session )
```

Reads the information stored within the configuration file for a single session.

Parameters

<i>int</i>	- the session id.
------------	-------------------

Returns

void

3.4.2.6 reset()

```
void Config::reset ( )
```

Discards all the changes made during a session.

Returns

void

3.4.2.7 save()

```
void Config::save ( )
```

Saves the information of the current session within the configuration file.

Returns

void

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

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

3.5 Effects Class Reference

The effects module. Currently unsupported.

```
#include <effects.h>
```

Public Member Functions

- [Effects](#) ()
Class constructor.
- void [initialize_effects](#) (const bool[F_NUM_INPUTS][NUM_EFFECTS])
Implicit initialization of all effects for each channel, depending on the state of the current session.
- bool [toggle_effect](#) (int, int)
Enables/Disables the an effect.
- void [apply](#) (float *[F_NUM_INPUTS])
Function to apply all the enabled effects to the input signal altogether. Currently unsupported.

3.5.1 Detailed Description

The effects module. Currently unsupported.

3.5.2 Member Function Documentation

3.5.2.1 apply()

```
void Effects::apply (
    float * input_buffers[F_NUM_INPUTS] )
```

Function to apply all the enabled effects to the input signal altogether. Currently unsupported.

Parameters

<i>float*[F_NUM_INPU↵TS]</i>	- the input signal
------------------------------	--------------------

Returns

void

3.5.2.2 initialize_effects()

```
void Effects::initialize_effects (
    const bool effects_curr_state[F_NUM_INPUTS][NUM_EFFECTS] )
```

Implicit initialization of all effects for each channel, depending on the state of the current session.

Parameters

<i>const</i>	<code>bool[F_NUM_INPUTS][NUM_EFFECTS]</code> - a boolean table containing the states of the effects (enabled/disabled) for each input channel.
--------------	--

Returns

void

3.5.2.3 toggle_effect()

```
bool Effects::toggle_effect (
    int ch,
    int eff )
```

Enables/Disables the an effect.

Parameters

<i>int</i>	- the corresponding channel
<i>int</i>	- the corresponding effect

Returns

bool - the current state of the corresponding effect after processing.

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

- include/effects.h
- src/effects.cpp

3.6 Handshake Class Reference

The jack-audio client.

```
#include <handshake.h>
```

Public Member Functions

- [Handshake](#) ()
Class constructor.
- void [link_client](#) (char *)
Function to open the jack client.
- void [set_process_callback](#) ()
Function to set the callback function used for streaming the audio signal.
- void [prevent_failure](#) ()
Function to set another function to be called in case the system fails.
- void [register_input_port](#) (int)
Function to register a new input port.
- void [register_output_port](#) (int)
Function to register a new output port.
- void [activate](#) ()
Function to activate the jack client.
- void [register_devices](#) ()
Function to register the input and the output devices.

- void `set_buffer_size` ()
Function to change the buffer size using the preprocessor derivative used in [audio_settings.h](#).
- void `check_status` ()
Function to check the status of the client connection.
- void `disconnect_client` ()
Function to disconnect the client from the server.
- void `connect_input_device` (int)
Function to connect input ports with input devices.
- void `connect_output_device` (int)
Function to connect output ports with output devices.
- void `disconnect_input_device` (int)
Function to disconnect input ports from the connected input devices.
- void `disconnect_output_device` (int)
Function to disconnect output ports from the connected output devices.
- float * `get_input_buffer` (int)
Function to get the input buffer from a specific port-device.
- float * `get_output_buffer` (int)
Function to get the output buffer to which the output port-device is connected.
- void `get_input_buffer` (int, float *)
Function to get the input buffer from a specific port-device.
- void `get_output_buffer` (int, float *)
Function to get the output buffer to which the output port-device is connected.

3.6.1 Detailed Description

The jack-audio client.

3.6.2 Member Function Documentation

3.6.2.1 `activate()`

```
void Handshake::activate ( )
```

Function to activate the jack client.

Returns

void

3.6.2.2 `check_status()`

```
void Handshake::check_status ( )
```

Function to check the status of the client connection.

Returns

void

3.6.2.3 `connect_input_device()`

```
void Handshake::connect_input_device (
    int idx )
```

Function to connect input ports with input devices.

Parameters

<i>int</i>	- index of the input port.
------------	----------------------------

Returns

void

3.6.2.4 connect_output_device()

```
void Handshake::connect_output_device (
    int idx )
```

Function to connect output ports with output devices.

Parameters

<i>int</i>	- index of the output port.
------------	-----------------------------

Returns

void

3.6.2.5 disconnect_client()

```
void Handshake::disconnect_client ( )
```

Function to disconnect the client from the server.

Returns

void

3.6.2.6 disconnect_input_device()

```
void Handshake::disconnect_input_device (
    int idx )
```

Function to disconnect input ports from the connected input devices.

Parameters

<i>int</i>	- index of the input port.
------------	----------------------------

Returns

void

3.6.2.7 disconnect_output_device()

```
void Handshake::disconnect_output_device (
    int idx )
```

Function to disconnect output ports from the connected output devices.

Parameters

<i>int</i>	- index of the output port.
------------	-----------------------------

Returns

void

3.6.2.8 get_input_buffer() [1/2]

```
float * Handshake::get_input_buffer (
    int idx )
```

Function to get the input buffer from a specific port-device.

Parameters

<i>int</i>	- index of the input port.
------------	----------------------------

Returns

float* - the input buffer

3.6.2.9 get_input_buffer() [2/2]

```
void Handshake::get_input_buffer (
    int ,
    float * )
```

Function to get the input buffer from a specific port-device.

Parameters

<i>int</i>	- index of the input port.
<i>float*</i>	- the input buffer passed by value.

Returns

void

3.6.2.10 get_output_buffer() [1/2]

```
float * Handshake::get_output_buffer (
    int idx )
```

Function to get the output buffer to which the output port-device is connected.

Parameters

<i>int</i>	- index of the output port.
------------	-----------------------------

Returns

float* - the output buffer where the output signal is written to.

3.6.2.11 get_output_buffer() [2/2]

```
void Handshake::get_output_buffer (
    int ,
    float * )
```

Function to get the output buffer to which the output port-device is connected.

Parameters

<i>int</i>	- index of the output port.
<i>float*</i>	- the output buffer passed by value.

Returns

void

3.6.2.12 link_client()

```
void Handshake::link_client (
    char * name )
```

Function to open the jack client.

Parameters

<i>char*</i>	- the client name
--------------	-------------------

Returns

void

3.6.2.13 prevent_failure()

```
void Handshake::prevent_failure ( )
```

Function to set another function to be called in case the system fails.

Returns

void

3.6.2.14 register_devices()

```
void Handshake::register_devices ( )
```

Function to register the input and the output devices.

Returns

void

3.6.2.15 register_input_port()

```
void Handshake::register_input_port (
    int idx )
```

Function to register a new input port.

Parameters

<i>int</i>	- index of the input port.
------------	----------------------------

Returns

void

3.6.2.16 register_output_port()

```
void Handshake::register_output_port (
    int idx )
```

Function to register a new output port.

Parameters

<i>int</i>	- index of the output port.
------------	-----------------------------

Returns

void

3.6.2.17 set_buffer_size()

```
void Handshake::set_buffer_size ( )
```

Function to change the buffer size using the preprocessor derivative used in [audio_settings.h](#).

Returns

void

3.6.2.18 set_process_callback()

```
void Handshake::set_process_callback ( )
```

Function to set the callback function used for streaming the audio signal.

Returns

void

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

- include/handshake.h
- src/handshake.cpp

3.7 hardwareInterface Class Reference

The generic interface of the pi-loop. It has an input and an output interface.

```
#include <interface.h>
```

Public Member Functions

- [hardwareInterface](#) ()
Class constructor.

- void `get_metro_display` (int)
Method to call the metronome display function triggered by the [Metronome](#).
- void `get_display_initializer` (DisplayInit)
Method to initialize the display for some display components (i.e. if EFF1 is enabled by default on the current session)
- void `listen` (void(PiLoop::*_notify)(Trigger), PiLoop &)
Prime method that runs on the ui thread.
- void `display` ()
Prime method that runs on the display thread.
- void `update` (Response)
Method to update the display thread upon user interventions.
- void `deafen` ()
Method to shutdown the interface.

3.7.1 Detailed Description

The generic interface of the pi-loop. It has an input and an output interface.

3.7.2 Member Function Documentation

3.7.2.1 deafen()

```
void hardwareInterface::deafen ( )
```

Method to shutdown the interface.

Returns

void

3.7.2.2 display()

```
void hardwareInterface::display ( )
```

Prime method that runs on the display thread.

Returns

void

3.7.2.3 get_display_initializer()

```
void hardwareInterface::get_display_initializer (
    DisplayInit data )
```

Method to initialize the display for some display components (i.e. if EFF1 is enabled by default on the current session)

Parameters

<code>int[9]</code>	- integer data to initialize the display object.
---------------------	--

Returns

void

3.7.2.4 get_metro_display()

```
void hardwareInterface::get_metro_display (
    int state )
```

Method to call the metronome display function triggered by the [Metronome](#).

Parameters

<i>int</i>	- the metronome intonation signal.
------------	------------------------------------

Returns

void

3.7.2.5 listen()

```
void hardwareInterface::listen (
    void(PiLoop::*)(Trigger) _notify,
    PiLoop & pl )
```

Prime method that runs on the ui thread.

Parameters

<i>void</i>	(PiLoop::*_notify)(Trigger) - the function pointer to the PiLoop class that is responsible for receiving the trigger from the current function.
<i>PiLoop&</i>	- reference to the PiLoop object.

Returns

void

3.7.2.6 update()

```
void hardwareInterface::update (
    Response response )
```

Method to update the display thread upon user interventions.

Parameters

<i>Response</i>	- post-process signal to update the display thread.
-----------------	---

Returns

void

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

- include/interface.h
- src/interface.cpp

3.8 Keyboard Class Reference

The keyboard as the computer-based input interface.

```
#include <keyboard.h>
```

Public Member Functions

- [Keyboard](#) ()
Class constructor.
- void [is_pressed](#) ([Trigger](#) &)
Function to return if a keyboard input is pressed by the user.

3.8.1 Detailed Description

The keyboard as the computer-based input interface.

3.8.2 Member Function Documentation

3.8.2.1 is_pressed()

```
void Keyboard::is_pressed (
    Trigger & trigger )
```

Function to return if a keyboard input is pressed by the user.

Parameters

<i>Trigger</i> &	- the trigger to be returned.
------------------	-------------------------------

Returns

void

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

- include/computer/keyboard.h
- src/computer/keyboard.cpp

3.9 Leds Class Reference

LEDs as the GPIO-based output interface.

```
#include <leds.h>
```

Public Member Functions

- [Leds](#) ()
Class constructor.
- void [initialize_leds](#) ([DisplayInit](#))
Method to implicitly initialize some of the output display components.
- void [display](#) ()
Threaded function running on the display thread.
- void [perform_operation](#) ([Response](#))
Function running on the session thread, and carries the system's response to the displayer.
- void [set_metro_state](#) (int)
Method to set the metronome's state.
- void [tick_tock](#) ()
Method to display the metronome's state.
- void [turnOff](#) ()
Method to shutdown the displayer.

3.9.1 Detailed Description

LEDs as the GPIO-based output interface.

3.9.2 Member Function Documentation

3.9.2.1 display()

```
void Leds::display ( )
```

Threaded function running on the display thread.

Returns

void

3.9.2.2 initialize_leds()

```
void Leds::initialize_leds (
    DisplayInit states )
```

Method to implicitly initialize some of the output display components.

Parameters

<i>DisplayInit</i>	- effects, monitor states + the current session)
--------------------	--

Returns

void

3.9.2.3 perform_operation()

```
void Leds::perform_operation (
    Response response )
```

Function running on the session thread, and carries the system's response to the displayer.

Returns

void

3.9.2.4 set_metro_state()

```
void Leds::set_metro_state (
    int intonation )
```

Method to set the metronome's state.

Parameters

<i>int</i>	- the metronome's intonation signal
------------	-------------------------------------

Returns

void

3.9.2.5 tick_tock()

```
void Leds::tick_tock ( )
```

Method to display the metronome's state.

Returns

void

3.9.2.6 turnOff()

```
void Leds::turnOff ( )
```

Method to shutdown the displayer.

Returns

void

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

- include/gpio/leds.h
- src/gpio/leds.cpp

3.10 Looper Class Reference

The [Looper](#) class consisted of 3 loop-channels.

```
#include <looper.h>
```

Public Member Functions

- [Looper](#) ()
Class constructor.
- void [tap_alter_metronome](#) (bool)
Function to do the tap-tempo or to change the metronome to an alternative value of tempo and number of measures.
- LooperOutput * [update_buffer](#) (float *[F_NUM_INPUTS], bool[F_NUM_INPUTS])
Method that receives the input buffer signal from the monitor.
- void [recdub](#) (int, bool, [Response](#) &)
Function triggered by the REC button. Supports four (4) operations : record-dub/stop_recording/playback/erase↔ previous Depending on the current state of the looper, it decides which one of them to apply.
- void [stoperase](#) (int, bool, [Response](#) &)
Function triggered by the PLAY/STOP button. Supports three (3) operations : stop_playback/stop_recording/erase↔ _channel Depending on the current state of the looper, it decides which one of them to apply.
- void [start_stop_all](#) (bool, [Response](#) &)
Function triggered by the START_ALL button. Supports two operations : start_stop_all/ erase_all Depending on the current state of the looper, it decides which one of them to apply.
- void [volume_change](#) (int, int)
Sets a channel's volume with a specific value.
- void [volume_up](#) (int)
Increase the channel's volume by a channels-defined volume step. Used for when changing volumes with keyboard keys.
- void [volume_down](#) (int)
Decrease the channel's volume by a channels-defined volume step. Used for when changing volumes with keyboard keys.
- bool [save](#) ()
Save the looper state. Unsupported yet.
- void [display_states](#) ()

Display the looper's current state. Used for debugging.

- int `get_metronome_state` ()

Function used to update the metronome's display status.

- void `reset` ()

Resets both the looper and the channels attached to it to the industrial conditions! (i.e. used for when changing session)

3.10.1 Detailed Description

The `Looper` class consisted of 3 loop-channels.

3.10.2 Member Function Documentation

3.10.2.1 `display_states()`

```
void Looper::display_states ( )
```

Display the looper's current state. Used for debugging.

Returns

void

3.10.2.2 `get_metronome_state()`

```
int Looper::get_metronome_state ( )
```

Function used to update the metronome's display status.

Returns

int - the intonation value of the metronome.

3.10.2.3 `recdub()`

```
void Looper::recdub (
    int channel,
    bool isHold,
    Response & response )
```

Function triggered by the REC button. Supports four (4) operations : record-dub/stop_recording/playback/erase_↔ previous Depending on the current state of the looper, it decides which one of them to apply.

Parameters

<i>int</i>	- the related channel
<i>bool</i>	- whether the button was pressed consecutively or not. If yes, it is an undub signal. Otherwise it is one among the rest of the operations.
<i>Response&</i>	- the output response that is used to update the display state of the looper.

Returns

void

3.10.2.4 reset()

```
void Looper::reset ( )
```

Resets both the looper and the channels attached to it to the industrial conditions! (i.e. used for when changing session)

Returns

void

3.10.2.5 save()

```
bool Looper::save ( )
```

Save the looper state. Unsupported yet.

Returns

void

3.10.2.6 start_stop_all()

```
void Looper::start_stop_all (
    bool isHold,
    Response & response )
```

Function triggered by the START_ALL button. Supports two operations : start_stop_all/ erase_all Depending on the current state of the looper, it decides which one of them to apply.

Parameters

<i>bool</i>	- whether the button was pressed consecutively or not. If yes, it is an erase-all-channels signal. Otherwise it is a signal that aims to pause all channels or to start playing all channels
<i>Response&</i>	- the output response that is used to update the display state of the looper.

Returns

void

3.10.2.7 stoperase()

```
void Looper::stoperase (
    int channel,
    bool isHold,
    Response & response )
```

Function triggered by the PLAY/STOP button. Supports three (3) operations : stop_playback/stop_↔ recording/erase_channel Depending on the current state of the looper, it decides which one of them to apply.

Parameters

<i>int</i>	- the related channel
<i>bool</i>	- whether the button was pressed consecutively or not. If yes, it is an erase signal. Otherwise it is one among the rest of the operations.

Parameters

<i>Response&</i>	- the output response that is used to update the display state of the looper.
----------------------	---

Returns

void

3.10.2.8 tap_alter_metronome()

```
void Looper::tap_alter_metronome (
    bool isHold )
```

Function to do the tap-tempo or to change the metronome to an alternative value of tempo and number of measures.

Parameters

<i>bool</i>	- if the button is pressed consecutively. if true, then it alters the tempo. Otherwise it is a tap-tempo signal.
-------------	--

Returns

void

3.10.2.9 update_buffer()

```
LooperOutput * Looper::update_buffer (
    float * input[F_NUM_INPUTS],
    bool armEnabled[F_NUM_INPUTS] )
```

Method that receives the input buffer signal from the monitor.

Parameters

<i>float*[F_NUM_INPUTS]</i>	- the input buffer sent by the monitor.
<i>bool[F_NUM_INPUTS]</i>	- the arm states

Returns

`std::array< std::array<float, BUFFER_SIZE>, F_NUM_OUTPUTS>` - the looper output buffer

3.10.2.10 volume_change()

```
void Looper::volume_change (
    int channel,
    int volume )
```

Sets a channel's volume with a specific value.

Parameters

<i>int</i>	- the corresponding channel
<i>int</i>	- the value of the volume as type of integer percentage.

Returns

void

3.10.2.11 volume_down()

```
void Looper::volume_down (
    int channel )
```

Decrease the channel's volume by a channels-defined volume step. Used for when changing volumes with keyboard keys.

Parameters

<i>int</i>	- the corresponding channel
------------	-----------------------------

Returns

void

3.10.2.12 volume_up()

```
void Looper::volume_up (
    int channel )
```

Increase the channel's volume by a channels-defined volume step. Used for when changing volumes with keyboard keys.

Parameters

<i>int</i>	- the corresponding channel
------------	-----------------------------

Returns

void

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

- include/looper.h
- src/looper.cpp

3.11 LooperState Struct Reference

Stores the looper state.

```
#include <response.h>
```

Public Member Functions

- [LooperState](#) ()
Default constructor.
- [LooperState](#) (const [LooperState](#) &obj)
Copy constructor.
- [LooperState](#) & [operator=](#) (const [LooperState](#) &obj)
Assignment operator.
- void [reset](#) ()
Reset function resets the object member variables to the default values.

Public Attributes

- `std::atomic< bool > playbacks` [3]
- `std::atomic< bool > record` [3]
- `std::atomic< int > num_tracks` [3]
- `std::atomic< int > is_changed`

3.11.1 Detailed Description

Stores the looper state.

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

- `include/response.h`

3.12 Menu Class Reference

The programs menu that acts as the session manager.

```
#include <menu.h>
```

Public Member Functions

- `Menu ()`
Class constructor.
- `void setup ()`
Function to apply setup actions after initialization and before running.
- `void load_session ()`
Prime function to run on the session thread.
- `void notify_menu (Trigger, Response &)`
Function to receive a trigger message and return a response message .
- `void unload ()`
Function to shutdown menu.
- `void set_display_initializer (std::function< void(DisplayInit)>)`
Function that carries a function pointer to the session obj.
- `void set_metro_display (std::function< void(int)>)`
Function that carries a function pointer to the session obj.

3.12.1 Detailed Description

The programs menu that acts as the session manager.

3.12.2 Member Function Documentation

3.12.2.1 `load_session()`

```
void Menu::load_session ( )
```

Prime function to run on the session thread.

Returns

`void`

3.12.2.2 notify_menu()

```
void Menu::notify_menu (
    Trigger trigger,
    Response & response )
```

Function to receive a trigger message and return a response message .

Parameters

<i>Trigger</i>	- the user input signal
<i>Response&</i>	- the response signal (passed by reference) to reach the display methods.

Returns

void

3.12.2.3 set_display_initializer()

```
void Menu::set_display_initializer (
    std::function< void(DisplayInit)> f )
```

Function that carries a function pointer to the session obj.

Parameters

<i>std::function< void(int[9])></i>	- the function to initialize the display object with info obtained by the session object.
---	---

Returns

void

3.12.2.4 set_metro_display()

```
void Menu::set_metro_display (
    std::function< void(int)> f )
```

Function that carries a function pointer to the session obj.

Parameters

<i>std::function< void(int)></i>	- the function that is going to be called each time the metronomes ticks.
--	---

Returns

void

3.12.2.5 setup()

```
void Menu::setup ( )
```

Function to apply setup actions after initialization and before running.

Returns

void

3.12.2.6 unload()

```
void Menu::unload ( )
```

Function to shutdown menu.

Returns

void

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

- include/menu.h
- src/menu.cpp

3.13 Metronome Class Reference

A metronome defined within [Looper](#).

```
#include <metronome.h>
```

Public Member Functions

- [Metronome](#) ()
Class constructor.
- void [lock](#) ()
function to set lock to true, so as to prevent metronome from changing.
- void [unlock](#) ()
function to set lock to false, so as to enable resetting of the metronome.
- void [pause](#) ()
function to pause the counting of the metronome.
- void [unpause](#) ()
function to unpause the metronome.
- void [start_timing](#) ()
function to start timing the first loop.
- void [stop_timing](#) (int)
function to stop timing and set the tempo.
- void [tap_tempo](#) ()
function to read taps from the user for setting the tempo
- int [tick_tock](#) ()
function that counts the time and does the tick-tocking
- void [alter_tempo](#) ()
function that changes the tempo to the next most suitable form.
- void [clear](#) ()
Function used to reset the metronome.(i.e. for when changing session)
- void [sync](#) ()
Helper function to synchronize metronome with the looper.

3.13.1 Detailed Description

A metronome defined within [Looper](#).

3.13.2 Member Function Documentation

3.13.2.1 alter_tempo()

```
void Metronome::alter_tempo ( )
```

function that changes the tempo to the next most suitable form.

Returns

void

3.13.2.2 clear()

```
void Metronome::clear ( )
```

Function used to reset the metronome.(i.e. for when changing session)

Returns

void

3.13.2.3 lock()

```
void Metronome::lock ( )
```

function to set lock to true, so as to prevent metronome from changing.

Returns

void

3.13.2.4 pause()

```
void Metronome::pause ( )
```

function to pause the counting of the metronome.

Returns

void

3.13.2.5 start_timing()

```
void Metronome::start_timing ( )
```

function to start timing the first loop.

Returns

void

3.13.2.6 stop_timing()

```
void Metronome::stop_timing (
    int loop_length )
```

function to stop timing and set the tempo.

Parameters

<i>int</i>	- the loop length in samples.
------------	-------------------------------

Returns

void

3.13.2.7 sync()

```
void Metronome::sync ( )
```

Helper function to synchronize metronome with the looper.

Returns

void

3.13.2.8 tap_tempo()

```
void Metronome::tap_tempo ( )
```

function to read taps from the user for setting the tempo

Returns

void

3.13.2.9 tick_tock()

```
int Metronome::tick_tock ( )
```

function that counts the time and does the tick-tocking

Returns

int - the intonation of the tick. 0 if the first of the loop, 1 if first of the measure, 2 otherwise

3.13.2.10 unlock()

```
void Metronome::unlock ( )
```

function to set lock to false, so as to enable resetting of the metronome.

Returns

void

3.13.2.11 unpause()

```
void Metronome::unpause ( )
```

function to unpause the metronome.

Returns

void

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

- include/metronome.h
- src/metronome.cpp

3.14 Mixer Class Reference

The mixer module responsible to stream the output signal.

```
#include <mixer.h>
```

Public Member Functions

- [Mixer](#) ()
Class constructor.
- void [update_buffer](#) (float *[F_NUM_INPUTS], float *[F_NUM_OUTPUTS], LooperOutput &, bool[F_NUM_INPUTS])
Function that streams the output audio signal.
- void [save_jam](#) (std::string)
Function that saves the last minutes to a file.

3.14.1 Detailed Description

The mixer module responsible to stream the output signal.

3.14.2 Member Function Documentation

3.14.2.1 save_jam()

```
void Mixer::save_jam (
    std::string savepath )
```

Function that saves the last minutes to a file.

Parameters

<i>std::string</i>	- the path to save the file.
--------------------	------------------------------

Returns

void

3.14.2.2 update_buffer()

```
void Mixer::update_buffer (
    float * input_buffers[F_NUM_INPUTS],
    float * output_buffers[F_NUM_OUTPUTS],
    LooperOutput & looper_buffers,
    bool monitorIn[F_NUM_INPUTS] )
```

Function that streams the output audio signal.

Parameters

<i>float*[F_NUM_INPUTS]</i>	- the monitor audio input signal
<i>float*[F_NUM_OUTPUTS]</i>	- the output buffers connected to the speakers
<i>LooperOutput</i>	- the looper output buffers
<i>bool[F_NUM_INPUTS]</i>	- bool array to indicate if monitorIn is enabled.

Returns

void

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

- include/mixer.h
- src/mixer.cpp

3.15 Monitor Class Reference

The monitor module encapsulates the jack audio client.

```
#include <monitor.h>
```

Public Member Functions

- [Monitor](#) ()
Class constructor.
- void [connect](#) (char *)
Function to connect the audio client to the audio server.
- void [disconnect](#) ()
Function to disconnect the jack audio client from the audio server.
- void [initialize_effects](#) (const bool[F_NUM_INPUTS][NUM_EFFECTS])
Function to initialize the effects (enabled/disabled) depending on the state of the session when last saved.
- bool [toggle_effect](#) (int, int)
Function to change state of effect.
- void [mute_input](#) (int)
Function to mute the input signal.
- void [mute_output](#) (int)
Function to mute the output signal.
- int [process](#) ()
The callback function used to retrieve the input buffers.
- void [set_stream_buffer](#) (std::function< void(float *[F_NUM_INPUTS], float *[F_NUM_OUTPUTS])>)
Function to set a member function with a function pointer to a non member function from another class so as to have the incoming buffers updated outside of the class scope.
- bool [toggle_states](#) (bool, int)
Function to change the state of the monitor options (ARM/MNTR)
- void [update_states](#) (bool, int, bool)
Function to set the state of the monitor options (ARM/MNTR) with a specific value. Used for initialization.
- void [get_states](#) (bool[F_NUM_INPUTS], bool)
Function to get the current state of the monitor options (ARM/MNTR).

3.15.1 Detailed Description

The monitor module encapsulates the jack audio client.

3.15.2 Member Function Documentation

3.15.2.1 connect()

```
void Monitor::connect (
    char * name )
```

Function to connect the audio client to the audio server.

Parameters

<i>char*</i>	- the client name
--------------	-------------------

Returns

void

3.15.2.2 disconnect()

```
void Monitor::disconnect ( )
```

Function to disconnect the jack audio client from the audio server.

Returns

void

3.15.2.3 get_states()

```
void Monitor::get_states (
    bool states[F_NUM_INPUTS],
    bool is_mntr )
```

Function to get the current state of the monitor options (ARM/MNTR).

Parameters

<i>bool[F_NUM_INPUTS]</i>	- the values to be set.
<i>bool</i>	- if it is MNTR or ARM.

Returns

void

3.15.2.4 initialize_effects()

```
void Monitor::initialize_effects (
    const bool effects_curr_state[F_NUM_INPUTS][NUM_EFFECTS] )
```

Function to initialize the effects (enabled/disabled) depending on the state of the session when last saved.

Parameters

<i>const</i>	<i>bool[F_NUM_INPUTS][NUM_EFFECTS]</i> - a table of boolean to declare the status (enabled/disabled) of each effect for each input channel
--------------	--

Returns

void

3.15.2.5 mute_input()

```
void Monitor::mute_input (
    int device )
```

Function to mute the input signal.

Parameters

<i>int</i>	- refers to the corresponding channel
------------	---------------------------------------

Returns

void

3.15.2.6 mute_output()

```
void Monitor::mute_output (
    int device )
```

Function to mute the output signal.

Parameters

<i>int</i>	- refers to the corresponding output channel
------------	--

Returns

void

3.15.2.7 process()

```
int Monitor::process ( )
```

The callback function used to retrieve the input buffers.

Returns

int - zero on success, non-zero on error

The process callback for this JACK application is called in a special realtime thread once for each audio cycle.

3.15.2.8 set_stream_buffer()

```
void Monitor::set_stream_buffer (
    std::function< void(float *[F_NUM_INPUTS], float *[F_NUM_OUTPUTS])> )
```

Function to set a member function with a function pointer to a non member function from another class so as to have the incoming buffers updated outside of the class scope.

Parameters

<i>std::function<void(float</i>	<i>*[F_NUM_INPUTS],float *[F_NUM_OUTPUTS])></i> - the pointer to the non-member function.
------------------------------------	--

Returns

void

3.15.2.9 toggle_effect()

```
bool Monitor::toggle_effect (
```

```
int ch,
int eff )
```

Function to change state of effect.

Parameters

<i>int</i>	- refers to the corresponding channel
<i>int</i>	- refers to the corresponding effect

Returns

void

3.15.2.10 toggle_states()

```
bool Monitor::toggle_states (
    bool is_mntr,
    int channel )
```

Function to change the state of the monitor options (ARM/MNTR)

Parameters

<i>bool</i>	- if it is MNTR or ARM.
<i>int</i>	- the corresponding channel

Returns

bool - the current state of the monitor-button.

3.15.2.11 update_states()

```
void Monitor::update_states (
    bool is_mntr,
    int input_channel,
    bool val )
```

Function to set the state of the monitor options (ARM/MNTR) with a specific value. Used for initialization.

Parameters

<i>bool</i>	- if it is MNTR or ARM.
<i>int</i>	- the corresponding channel
<i>bool</i>	- the value to be set.

Returns

void

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

- include/monitor.h
- src/monitor.cpp

3.16 PiLoop Class Reference

The threading manager. It starts the program.


```
#include <piloop.h>
```

Public Member Functions

- [PiLoop](#) ()
Class constructor. Responsible for setting up the threads and some function pointers to callback functions.
- void [start_console](#) ()
Runs all the four threads.

3.16.1 Detailed Description

The threading manager. It starts the program.

3.16.2 Member Function Documentation

3.16.2.1 start_console()

```
void PiLoop::start_console ( )
```

Runs all the four threads.

Returns

void

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

- include/piloop.h
- src/piloop.cpp

3.17 Potentiometers Class Reference

[Potentiometers](#) to adjust the looper-channel's volume.

```
#include <potentiometers.h>
```

Public Member Functions

- [Potentiometers](#) ()
Class constructor.
- void [setup](#) (int)
Method used as an implicit constructor to setup the instance of the class.
- void [is_changed](#) ([Trigger](#) &)
Function to notify a knob is been rotated or not.

3.17.1 Detailed Description

[Potentiometers](#) to adjust the looper-channel's volume.

3.17.2 Member Function Documentation

3.17.2.1 is_changed()

```
void Potentiometers::is_changed (
    Trigger & trigger )
```

Function to notify a knob is been rotated or not.

Parameters

<i>Trigger&</i>	- the trigger to be returned.
---------------------	-------------------------------

Returns

void

3.17.2.2 setup()

```
void Potentiometers::setup (
    int base )
```

Method used as an implicit constructor to setup the instance of the class.

Parameters

<i>int</i>	- the base number for the mcp3008 ADC
------------	---------------------------------------

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

- include/gpio/potentiometers.h
- src/gpio/potentiometers.cpp

3.18 Response Struct Reference

Defines a response message destined to the displayer module of the program.

```
#include <response.h>
```

Public Member Functions

- [Response](#) ()
Default constructor.
- [Response](#) (const [Response](#) &obj)
Copy constructor.
- [Response](#) & [operator=](#) (const [Response](#) &obj)
Assignment operator.
- bool [isEmpty](#) ()
Function that returns if a [Response](#) message is empty.
- void [reset](#) ()
Reset function resets the object member variables to the default values.

Public Attributes

- std::atomic< int > **holder** {-1}
- std::atomic< Message > **msg** {EMPTY}
- std::atomic< int > **value** {-1}
- [LooperState](#) **looper_state**

3.18.1 Detailed Description

Defines a response message destined to the displayer module of the program.

3.18.2 Member Function Documentation

3.18.2.1 isEmpty()

```
bool Response::isEmpty ( ) [inline]
```

Function that returns if a [Response](#) message is empty.

Returns

bool - True if is empty (if msg == EMPTY or if looper_state is not changed). Otherwise false

3.18.2.2 reset()

```
void Response::reset ( ) [inline]
```

Reset function resets the object member variables to the default values.

Returns

void

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

- include/response.h

3.19 Screen Class Reference

The screen as the computer-based output interface.

```
#include <screen.h>
```

Public Member Functions

- [Screen](#) ()
Class constructor.
- void [initialize_states](#) (DisplayInit)
Method to implicitly initialize some output display components.
- void [display](#) ()
Threaded function running on the display thread.
- void [perform_operation](#) ([Response](#))
Function running on the session thread, and carries the system's response to the displayer.
- void [tick_tock](#) (int)
Method to display the metronome's state.
- void [turnOff](#) ()
Method to shutdown the displayer.

3.19.1 Detailed Description

The screen as the computer-based output interface.

3.19.2 Member Function Documentation

3.19.2.1 display()

```
void Screen::display ( )
```

Threaded function running on the display thread.

Returns

void

3.19.2.2 initialize_states()

```
void Screen::initialize_states (
    DisplayInit comp_states )
```

Method to implicitly initialize some output display components.

Parameters

<i>DisplayInit</i>	- effects, monitor states + the current session)
--------------------	--

Returns

void

3.19.2.3 perform_operation()

```
void Screen::perform_operation (
    Response response )
```

Function running on the session thread, and carries the system's response to the displayer.

Returns

void

3.19.2.4 tick_tock()

```
void Screen::tick_tock (
    int intonation )
```

Method to display the metronome's state.

Parameters

<i>int</i>	- the metronome's intonation signal
------------	-------------------------------------

Returns

void

3.19.2.5 turnOff()

```
void Screen::turnOff ( )
```

Method to shutdown the displayer.

Returns

void

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

- include/computer/screen.h
- src/computer/screen.cpp

3.20 Session Class Reference

Defines a musical session. It is consisted of a [Monitor](#), a [Looper](#) and a [Mixer](#).

```
#include <session.h>
```

Public Member Functions

- [Session](#) ()
Class constructor.
- void [setup](#) ()
Setup whatever needed for implicit initialization of the imported classes.
- void [load](#) ()
Threaded function. This one is to have the audio client up and running.
- void [save](#) ()
Function to save the current session. Currently it is partially unsupported ([Looper](#) session will not be saved).
- void [migrate](#) (int)
Function used to change session. @params int - the next session.
- void [evacuate](#) ()
Function shutdown the session.
- void [notify_session](#) (Trigger, Response &)
Function used to retrieve the user's input trigger, and process the message and pass it all around the program, getting sure that a response will be returned for display.
- void [save_jam](#) ()
Function to save the last minutes of the current jam.
- void [set_name](#) (const char *)
Function to set the name of the current session.
- const char * [get_name](#) ()
Function to return the name of the current session.
- void [set_metronome_display](#) (std::function< void(int)>)
Function to pass the metronome-display-function-pointer.
- void [set_disp_initializer](#) (std::function< void(DisplayInit)>)
Function that gets a function pointer to set a member function, which is used to implicitly initialize the display module with default/predefined values.

3.20.1 Detailed Description

Defines a musical session. It is consisted of a [Monitor](#), a [Looper](#) and a [Mixer](#).

3.20.2 Member Function Documentation

3.20.2.1 [evacuate\(\)](#)

```
void Session::evacuate ( )
```

Function shutdown the session.

Returns

void

3.20.2.2 [get_name\(\)](#)

```
const char * Session::get_name ( )
```

Function to return the name of the current session.

Returns

const char* - the name of the current session.

3.20.2.3 load()

```
void Session::load ( )
```

Threaded function. This one is to have the audio client up and running.

Returns

void

3.20.2.4 migrate()

```
void Session::migrate (
    int next_session )
```

Function used to change session. @params int - the next session.

Returns

void

3.20.2.5 notify_session()

```
void Session::notify_session (
    Trigger trigger,
    Response & response )
```

Function used to retrieve the user's input trigger, and process the message and pass it all around the program, getting sure that a response will be returned for display.

Returns

void

3.20.2.6 save()

```
void Session::save ( )
```

Function to save the current session. Currently it is partially unsupported ([Looper](#) session will not be saved).

Returns

void

3.20.2.7 save_jam()

```
void Session::save_jam ( )
```

Function to save the last minutes of the current jam.

Returns

void

3.20.2.8 set_disp_initializer()

```
void Session::set_disp_initializer (
    std::function< void(DisplayInit)> di )
```

Function that gets a function pointer to set a member function, which is used to implicitly initialize the display module with default/predefined values.

Parameters

<code>std::function<void(int[9])></code>	- the pointer to the function that takes 9 integer values to initialize the output states before use.
--	---

Returns

void

3.20.2.9 set_metronome_display()

```
void Session::set_metronome_display (
    std::function< void(int)> metrodisp )
```

Function to pass the metronome-display-function-pointer.

Parameters

<code>std::function<void(int)></code>	- the metronome display function
---	----------------------------------

Returns

void

3.20.2.10 set_name()

```
void Session::set_name (
    const char * name )
```

Function to set the name of the current session.

Parameters

<code>const</code>	char* - the new name
--------------------	----------------------

Returns

void

3.20.2.11 setup()

```
void Session::setup ( )
```

Setup whatever needed for implicit initialization of the imported classes.

Returns

void

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

- include/session.h
- src/session.cpp

3.21 time_signature Struct Reference

Struct to hold the information of rythm.

```
#include <metronome.h>
```

Public Attributes

- int **numerator**
- int **denominator**

3.21.1 Detailed Description

Struct to hold the information of rythm.

Parameters

<i>int</i>	numerator - the rythm numerator
<i>int</i>	denominator - the rythm denominator

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

- include/metronome.h

3.22 Trigger Struct Reference

Holds the triggers of the user.

```
#include <trigger.h>
```

Public Member Functions

- [Trigger](#) ()
Default constructor.
- [Trigger](#) (const [Trigger](#) &obj)
Copy constructor.
- [Trigger](#) & **operator=** (const [Trigger](#) &obj)
Assignment operator.
- void [reset](#) ()
Reset function resets the object member variables to the default values.

Public Attributes

- std::atomic< Control > **control** {IEMPTY}
- std::atomic< int > **subval** {-1}

3.22.1 Detailed Description

Holds the triggers of the user.

3.22.2 Member Function Documentation

3.22.2.1 reset()

```
void Trigger::reset ( ) [inline]
```

Reset function resets the object member variables to the default values.

Returns

void

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

- include/trigger.h

3.23 UI Class Reference

Defines a computer based interface consisted of a keyboard and the screen.

```
#include <computer.h>
```

Public Member Functions

- [UI](#) ()
Class constructor.
- void [listen_user](#) ([Trigger](#) &)
Method to pass the trigger stucture by reference to a function that gets the keyboard input messages by the user.
- void [update_output_state](#) ([Response](#))
Method to update the output based on the system's response message.
- void [show](#) ()
Method that runs on the display thread.
- void [initialize_display](#) (DisplayInit)
- void [metro_display](#) (int)
Method to call the displayer's tick_tock function.
- void [turnOff](#) ()
Method to shutdown UI.
- [UI](#) ()
Class constructor.
- void [listen_user](#) ([Trigger](#) &)
Method to pass the trigger stucture by reference to a function that gets the buttons triggers by the user.
- void [update_output_state](#) ([Response](#))
Method to update the output based on the system's response message.
- void [show](#) ()
Method that runs on the display thread.
- void [initialize_display](#) (DisplayInit)
- void [metro_display](#) (int)
Method to call the displayer's tick_tock function.
- void [turnOff](#) ()
Method to shutdown UI.

3.23.1 Detailed Description

Defines a computer based interface consisted of a keyboard and the screen.

Defines a GPIO based interface consisted of buttons, potentiometers and leds.

3.23.2 Member Function Documentation

3.23.2.1 initialize_display() [1/2]

```
void UI::initialize_display (
    DisplayInit data )
```

Method to pass some initialization values to the displayer before the program starts

Parameters

<i>DisplayInit</i>	- integer data to initialize the display object.
--------------------	--

Returns

void

3.23.2.2 initialize_display() [2/2]

```
void UI::initialize_display (
    DisplayInit )
```

Method to pass some initialization values to the displayer before the program starts

Parameters

<i>DisplayInit</i>	- integer data to initialize the display object.
--------------------	--

Returns

void

3.23.2.3 listen_user() [1/2]

```
void UI::listen_user (
    Trigger & trigger )
```

Method to pass the trigger stucture by reference to a function that gets the keyboard input messages by the user.

Parameters

<i>Trigger&</i>	- The trigger to be set by the user using the pc-keyboard.
---------------------	--

Returns

void

3.23.2.4 listen_user() [2/2]

```
void UI::listen_user (
    Trigger & )
```

Method to pass the trigger stucture by reference to a function that gets the buttons triggers by the user.

Parameters

<i>Trigger&</i>	- The trigger to be set by the user using the GPIO-connected buttons.
---------------------	---

Returns

void

3.23.2.5 metro_display() [1/2]

```
void UI::metro_display (
    int state )
```

Method to call the displayer's tick_tock function.

Parameters

<i>int</i>	- the intonation signal obtained from the metronome obj.
------------	--

Returns

void

3.23.2.6 metro_display() [2/2]

```
void UI::metro_display (
    int )
```

Method to call the displayer's tick_tock function.

Parameters

<i>int</i>	- the intonation signal obtained from the metronome obj.
------------	--

Returns

void

3.23.2.7 show() [1/2]

```
void UI::show ( )
```

Method that runs on the display thread.

Returns

void

3.23.2.8 show() [2/2]

```
void UI::show ( )
```

Method that runs on the display thread.

Returns

void

3.23.2.9 turnOff() [1/2]

```
void UI::turnOff ( )
```

Method to shutdown [UI](#).

Returns

void

3.23.2.10 turnOff() [2/2]

```
void UI::turnOff ( )
```

Method to shutdown [UI](#).

Returns

void

3.23.2.11 update_output_state() [1/2]

```
void UI::update_output_state (
    Response response )
```

Method to update the output based on the system's response message.

Parameters

Response	- the program's response carried to the displayer.
--------------------------	--

Returns

void

3.23.2.12 update_output_state() [2/2]

```
void UI::update_output_state (
    Response )
```

Method to update the output based on the system's response message.

Parameters

Response	- the program's response carried to the displayer.
--------------------------	--

Returns

void

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

- include/computer.h
- include/gpio.h
- src/computer.cpp
- src/gpio.cpp

Index

- activate
 - Handshake, 17
- add_track
 - Channel, 9
- alter_tempo
 - Metronome, 33
- apply
 - Effects, 15
- AudioServer, 7
 - start, 7
 - stop, 7
- Buttons, 8
 - is_pressed, 8
 - setup, 8
- Channel, 8
 - add_track, 9
 - clean, 9
 - dub, 10
 - get_name, 10
 - get_num_tracks, 10
 - get_out_signal, 10
 - get_volume, 10
 - isEmpty, 11
 - rec, 11
 - reset, 11
 - reset_num_tracks, 11
 - set_name, 11
 - set_volume, 11
 - undub, 12
 - update_rec_signal, 12
 - volume_down, 12
 - volume_up, 12
- check_status
 - Handshake, 17
- clean
 - Channel, 9
- clear
 - Metronome, 34
- Config, 12
 - display, 13
 - get_button_state, 13
 - get_curr_session, 14
 - get_max_sessions, 14
 - open, 14
 - reset, 14
 - save, 14
- connect
 - Monitor, 37
- connect_input_device
 - Handshake, 17
- connect_output_device
 - Handshake, 18
- deafen
 - hardwareInterface, 22
- disconnect
 - Monitor, 38
- disconnect_client
 - Handshake, 18
- disconnect_input_device
 - Handshake, 18
- disconnect_output_device
 - Handshake, 18
- display
 - Config, 13
 - hardwareInterface, 22
 - Leds, 25
 - Screen, 43
- display_states
 - Looper, 27
- dub
 - Channel, 10
- Effects, 15
 - apply, 15
 - initialize_effects, 15
 - toggle_effect, 16
- evacuate
 - Session, 45
- get_button_state
 - Config, 13
- get_curr_session
 - Config, 14
- get_display_initializer
 - hardwareInterface, 22
- get_input_buffer
 - Handshake, 19
- get_max_sessions
 - Config, 14
- get_metro_display
 - hardwareInterface, 22
- get_metronome_state
 - Looper, 27
- get_name
 - Channel, 10
 - Session, 45
- get_num_tracks

- Channel, 10
- get_out_signal
 - Channel, 10
- get_output_buffer
 - Handshake, 19
- get_states
 - Monitor, 38
- get_volume
 - Channel, 10
- Handshake, 16
 - activate, 17
 - check_status, 17
 - connect_input_device, 17
 - connect_output_device, 18
 - disconnect_client, 18
 - disconnect_input_device, 18
 - disconnect_output_device, 18
 - get_input_buffer, 19
 - get_output_buffer, 19
 - link_client, 20
 - prevent_failure, 20
 - register_devices, 20
 - register_input_port, 20
 - register_output_port, 21
 - set_buffer_size, 21
 - set_process_callback, 21
- hardwareInterface, 21
 - deafen, 22
 - display, 22
 - get_display_initializer, 22
 - get_metro_display, 22
 - listen, 23
 - update, 23
- initialize_display
 - UI, 49, 50
- initialize_effects
 - Effects, 15
 - Monitor, 38
- initialize_leds
 - Leds, 25
- initialize_states
 - Screen, 43
- is_changed
 - Potentiometers, 41
- is_pressed
 - Buttons, 8
 - Keyboard, 24
- isEmpty
 - Channel, 11
 - Response, 42
- Keyboard, 23
 - is_pressed, 24
- Leds, 24
 - display, 25
 - initialize_leds, 25
 - perform_operation, 25
 - set_metro_state, 25
 - tick_tock, 25
 - turnOff, 26
- link_client
 - Handshake, 20
- listen
 - hardwareInterface, 23
- listen_user
 - UI, 50
- load
 - Session, 45
- load_session
 - Menu, 31
- lock
 - Metronome, 34
- Looper, 26
 - display_states, 27
 - get_metronome_state, 27
 - recdub, 27
 - reset, 28
 - save, 28
 - start_stop_all, 28
 - stoperase, 28
 - tap_alter_metronome, 29
 - update_buffer, 29
 - volume_change, 29
 - volume_down, 30
 - volume_up, 30
- LooperState, 30
- Menu, 31
 - load_session, 31
 - notify_menu, 31
 - set_display_initializer, 32
 - set_metro_display, 32
 - setup, 32
 - unload, 32
- metro_display
 - UI, 50, 51
- Metronome, 33
 - alter_tempo, 33
 - clear, 34
 - lock, 34
 - pause, 34
 - start_timing, 34
 - stop_timing, 34
 - sync, 35
 - tap_tempo, 35
 - tick_tock, 35
 - unlock, 35
 - unpause, 35
- migrate
 - Session, 46
- Mixer, 36
 - save_jam, 36
 - update_buffer, 36
- Monitor, 37
 - connect, 37

- disconnect, [38](#)
- get_states, [38](#)
- initialize_effects, [38](#)
- mute_input, [38](#)
- mute_output, [39](#)
- process, [39](#)
- set_stream_buffer, [39](#)
- toggle_effect, [39](#)
- toggle_states, [40](#)
- update_states, [40](#)
- mute_input
 - Monitor, [38](#)
- mute_output
 - Monitor, [39](#)
- notify_menu
 - Menu, [31](#)
- notify_session
 - Session, [46](#)
- open
 - Config, [14](#)
- pause
 - Metronome, [34](#)
- perform_operation
 - Leds, [25](#)
 - Screen, [44](#)
- PiLoop, [40](#)
 - start_console, [41](#)
- Potentiometers, [41](#)
 - is_changed, [41](#)
 - setup, [42](#)
- prevent_failure
 - Handshake, [20](#)
- process
 - Monitor, [39](#)
- rec
 - Channel, [11](#)
- recdub
 - Looper, [27](#)
- register_devices
 - Handshake, [20](#)
- register_input_port
 - Handshake, [20](#)
- register_output_port
 - Handshake, [21](#)
- reset
 - Channel, [11](#)
 - Config, [14](#)
 - Looper, [28](#)
 - Response, [43](#)
 - Trigger, [48](#)
- reset_num_tracks
 - Channel, [11](#)
- Response, [42](#)
 - isEmpty, [42](#)
 - reset, [43](#)
- save
 - Config, [14](#)
 - Looper, [28](#)
 - Session, [46](#)
- save_jam
 - Mixer, [36](#)
 - Session, [46](#)
- Screen, [43](#)
 - display, [43](#)
 - initialize_states, [43](#)
 - perform_operation, [44](#)
 - tick_tock, [44](#)
 - turnOff, [44](#)
- Session, [44](#)
 - evacuate, [45](#)
 - get_name, [45](#)
 - load, [45](#)
 - migrate, [46](#)
 - notify_session, [46](#)
 - save, [46](#)
 - save_jam, [46](#)
 - set_disp_initializer, [46](#)
 - set_metronome_display, [47](#)
 - set_name, [47](#)
 - setup, [47](#)
- set_buffer_size
 - Handshake, [21](#)
- set_disp_initializer
 - Session, [46](#)
- set_display_initializer
 - Menu, [32](#)
- set_metro_display
 - Menu, [32](#)
- set_metro_state
 - Leds, [25](#)
- set_metronome_display
 - Session, [47](#)
- set_name
 - Channel, [11](#)
 - Session, [47](#)
- set_process_callback
 - Handshake, [21](#)
- set_stream_buffer
 - Monitor, [39](#)
- set_volume
 - Channel, [11](#)
- setup
 - Buttons, [8](#)
 - Menu, [32](#)
 - Potentiometers, [42](#)
 - Session, [47](#)
- show
 - UI, [51](#)
- start
 - AudioServer, [7](#)
- start_console
 - PiLoop, [41](#)
- start_stop_all

- Looper, [28](#)
- start_timing
 - Metronome, [34](#)
- stop
 - AudioServer, [7](#)
- stop_timing
 - Metronome, [34](#)
- stoperase
 - Looper, [28](#)
- sync
 - Metronome, [35](#)
- tap_alter_metronome
 - Looper, [29](#)
- tap_tempo
 - Metronome, [35](#)
- tick_tock
 - Leds, [25](#)
 - Metronome, [35](#)
 - Screen, [44](#)
- time_signature, [47](#)
- toggle_effect
 - Effects, [16](#)
 - Monitor, [39](#)
- toggle_states
 - Monitor, [40](#)
- Trigger, [48](#)
 - reset, [48](#)
- turnOff
 - Leds, [26](#)
 - Screen, [44](#)
 - UI, [51](#)
- UI, [49](#)
 - initialize_display, [49](#), [50](#)
 - listen_user, [50](#)
 - metro_display, [50](#), [51](#)
 - show, [51](#)
 - turnOff, [51](#)
 - update_output_state, [52](#)
- undub
 - Channel, [12](#)
- unload
 - Menu, [32](#)
- unlock
 - Metronome, [35](#)
- unpause
 - Metronome, [35](#)
- update
 - hardwareInterface, [23](#)
- update_buffer
 - Looper, [29](#)
 - Mixer, [36](#)
- update_output_state
 - UI, [52](#)
- update_rec_signal
 - Channel, [12](#)
- update_states
 - Monitor, [40](#)
- volume_change
 - Looper, [29](#)
- volume_down
 - Channel, [12](#)
 - Looper, [30](#)
- volume_up
 - Channel, [12](#)
 - Looper, [30](#)