ospac

Generated by Doxygen 1.8.11

Contents

1 Ospac project documentation

1.1 Introduction

Ospac will take a multi-channel recording of a conversation and master this to a high-quality mix-down with support for intro and outro. It was developed due to the need of a batch solution for audio podcast creation. It is a rewrite and compilation of the scripts and methods used for the Modellansatz podcast (`http://modellansatz.de/` or `http://modellansatz.de/en`).

1.2 Concept

A main issue is to get things done without too much hassle. Therefore, the external dependencies are minimal: It depends on libsndfile only. Also, the channels are simply std::vectors of floats, therefore all is done in memory and unnecessary copy operations will take place. Furthermore, there is not yet an object oriented concept of generators, analyzers, filters, and consumers, and thus many filters are just static methods getting their thing done.

1.2.1 Fundamental classes

Single channels are represented by the Channel class. It consists of std::vector<float> and the corresponding bitrate. Multiple channels such as in stereo are represented by Channels, a std::vector<Channel>.

Loading and saving of Channels is done by the Wave class. It uses libsndfile to load wave files with an arbitrary number of channels and likewise saving of such channels.

All logging takes place using the Log class, that takes care of also showing the source code line number, run time and logging levels.

The actual command line interface is implemented in the OspacMain class. All defaults and actual workflows for audio processing are defined in there.

The conversion of physical quantities is done in the Physics class. So far, the only conversion is the distance of sound to time and vice versa with respect of the speed of sound at normal room conditions.

The Plot class delivers plain visualizations of Channels objects in netpbm format. All plots in this document were create by this functionality.

1.2.2 Single channel filter classes

In the following some of the active filter classes are illustrated. The examples given are in the test directory of the repository.

1.2.2.1 Leveling

The leveling is done in the SelectiveLeveler class: First of all, the filter identifies levels for silence, transition and signal. Then it tries to normalize the average l2 energy in a window of detected signals to a given energy level, while muting detected silence. The target energy level of transition sections is linearily dependent on their respective level. The actual muting or amplification is smoothened in a tolerance window to prevent continuous leveling.

This is an exemplary leveling result (top: original, bottom: result):

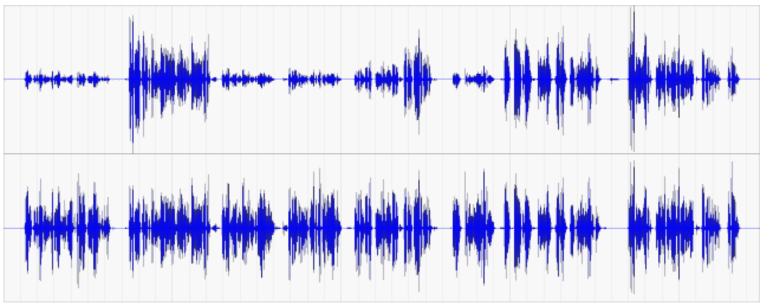


Figure 1 Result of selective leveler

1.2.2.2 Normalization and amplification

The Maximizer class deals with amplification and normalization of audio signals. Since amplification can easily lead to distortion it is always combined with a sigmoid limiter. (Of course this introduces distortion as well, but without wrap-arounds would occur leading to far worse clicking.)

Examples of amplification by factor 2 and 4 (top: original, below 6dB or 12dB attenuation):

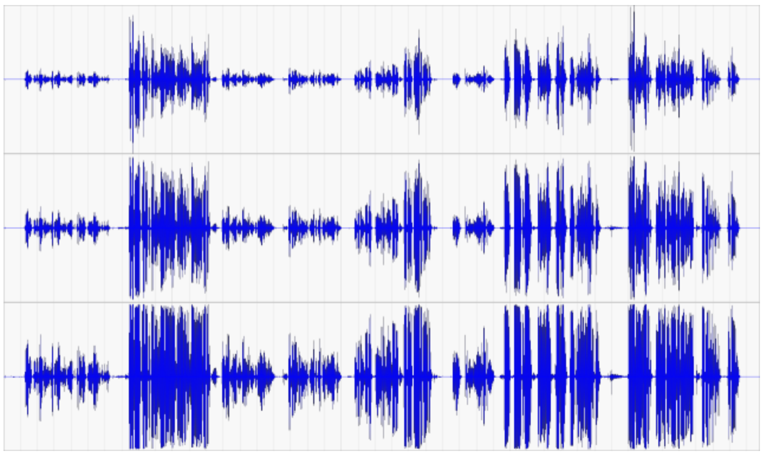


Figure 2 Result of amplification by 2 or 4

The normalization simply scales the signal to the full 16bit value range.

Example of normalization (top: original, bottom: result):

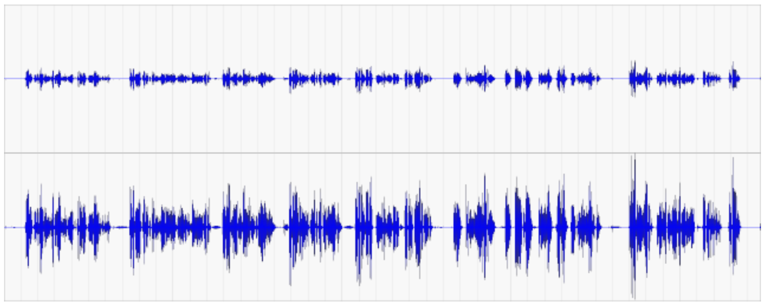


Figure 3 Result of normalization

1.2.2.3 Soft silence skipping

The Skip class offers a way to reduce long silent passages without influencing the natural way of speaking. First of all, this is done by only considering passages that have silence for longer than a certain time (0.5s) and then reducing only relatively to the time the silence extends longer. Therefore, longer pauses will remain longer pauses compared to others, but they will be shorter altogether.

Example of soft skipping (top: original, bottom: result):

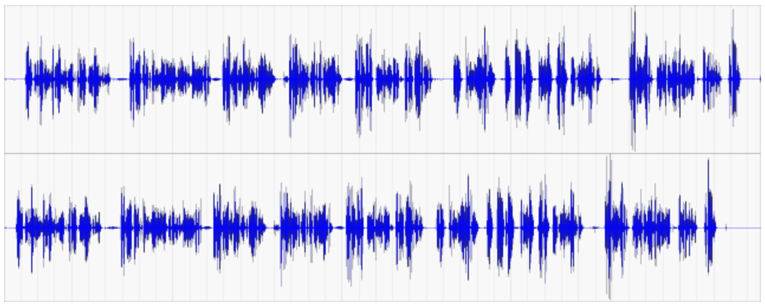


Figure 4 Result of soft skipping

1.2.3 Crosstalk filter classes

Crosstalk occurs when one microphone records signals from other channels as well. The following example shows two channels, where the second channel mainly consists of crosstalk of the first, and a small segment of original input.

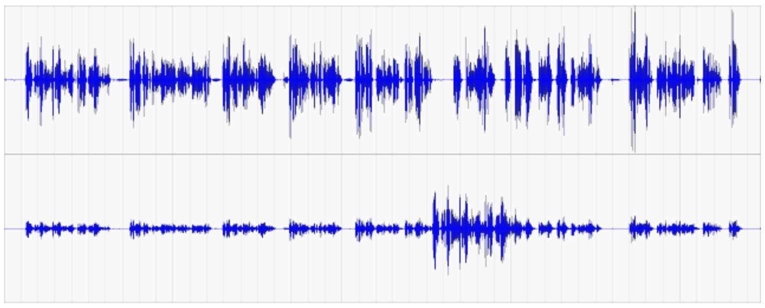


Figure 5 Exemplary two channel input

Ospac has two filters that deal with this issue. The traditional and robust crosstalk gate based on the activity on one channel compared to the others, and the experimental crosstalk filter that actively searches for crosstalk occurences.

1.2.3.1 Crosstalk gate

The CrosstalkGate computes activity levels on each channel and then mutes channels linearily with less activity compared to the current maximum activity. This filter was successfully applied to many podcast episodes.

The gate results in this output from above example:

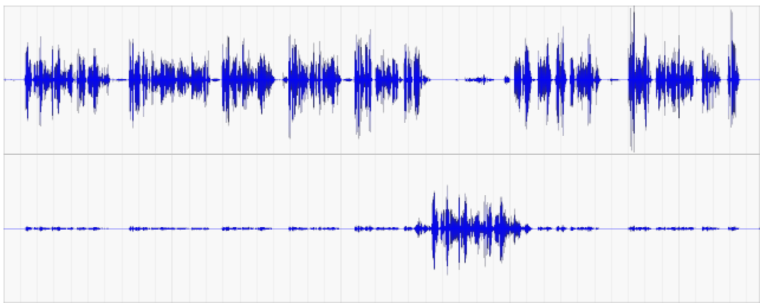


Figure 6 Result of crosstalk gate filter

1.2.3.2 Crosstalk filter

The CrosstalkFilter tries to actively identify crosstalk in other channels and mutes signals that mainly consist of crosstalk. The identification is done by scalar product on a comparison windows between the current signal and previous windows of other channels as a negative indicator, as well as the current signal and future crosstalk on other channels as a positive indicator for original input. This filter is a new development and should be considered experimental.

The filter results in this output from above example:

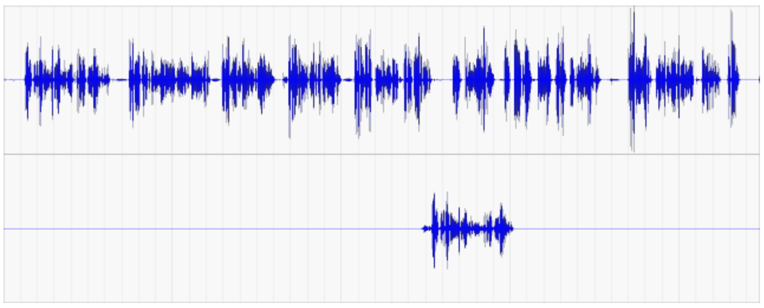


Figure 7 Result of crosstalk filter

1.2.4 Mix down filter

The Merge class merges two audio segments by either overlap or fading. In the overlap mode, both parts remain at full volume and should start and end with zero intensity. In the fading mode, the parts are linearly faded in and out in the transition phase. When the transition time is zero, the two parts are just joint after each other.

The MonoMix class simply superimposes all channels onto one mono channel. To prevent clipping this should be followed by normalization or amplification with sigmoid mapping.

The StereoMix class offers several modes of stereo mapping: In its simplest mode the various channels are mapped onto equidistant positions that are rendered with intensity differences onto the stereo channels.

A more sophisticated mode takes interaural delays into account yielding a far more realistic positioning of the speakers. This comes by cost of worse mono reproduction and decreased compressibility of the outcome.

Another mode takes occlusion of higher frequencies into account, this improves mono reproduction but currently the frequency bank filters are quite bad and therefore the outcome is improvable.

1.3 Plans for the future

Improved ospac user interface

The command line interface does not offer all capabilities the filters can offer, by far. For example, one cannot include pre-mixed channels for music and voice channels in the same part. Also the additional meta information could be included, but would make the calling parameters very long. Therefore, a settings file, based on a markup language, seems like a good idea to improve the user interface.

Object orientation

Design proper object oriented representation of concepts: Split analysis and filter methods in order to enable free combination of various concepts. And decrease the number of unnecessary copy operations.

More detailed merging parameters

At the moment, merging of parts can either be faded or overlapped on both sides. There are some cases, where more modes or asymmetric configurations could be helpful.

Direct invocation of encoders

Due to different command line interface of encoders, it might be a good idea to include the invocation of mp3, ogg, ... encoders into ospac. For this, the user interface should be improved due to the large number of meta tags that should be included in the interface.

Support for HRTF functions

As an additional 3d rendering solution the support for head related transfer functions (HRTF) would be highly interesting.

Synchronization of double enders

The skipping filter shows how channels can be shortened or enlenghted without perceivable change to the voice. At the same time, the crosstalk filter accurately matches signals from different channels on sample accuracy. Therefore everything is there to offer time dynamic synchronization of double enders that include low quality recordings of the other channels.

Multi threading support

Most of the filters can easily benefit from multi threading- either by parallel treatment of channels, or by time splitting in a channel for filters that do not have time-dependent side-effects.

Channel virtualization

To avoid excessive memory usage, the basic channel structure could be extended to support hard drive mapping of results. Only segments that are currently worked on are in memory and all other parts remain on disc.

2 Class Documentation

2.1 Channel Class Reference

Audio channel abstraction class.

```
#include <Channel.h>
```

Public Member Functions

- Channel ()
- Channel (unsigned rate)
- Channel (unsigned rate, const std::vector< float > &data)
- Channel (unsigned rate, unsigned size)
- float & operator[ ] (int)
- float operator[ ] (int) const
- unsigned size () const
- unsigned samplerate () const
- double l2norm () const
- double l2upnorm (float) const
- double l2downnorm (float) const
- double linfnorm () const
- Channel downsample (unsigned) const
- Channel downsampleEnergy (unsigned) const
- Channel resizeTo (unsigned) const
- Channel resampleTo (unsigned) const

2.1.1 Detailed Description

Audio channel abstraction class.

TODO: Switch to auto_ptr<float> to avoid copy operations at assignments. TODO: Virtualization of memory segments to avoid full memory operations.

2.1.2 Constructor & Destructor Documentation

2.1.2.1 Channel::Channel ( )

Create a new audio channel.

2.1.2.2 Channel::Channel (unsigned *rate*)

Create an audio channel with given sample rate

Parameters

rate	sample rate in Hertz (1/s)

2.1.2.3 Channel::Channel (unsigned *rate,* const std::vector< float > & *data*)

Create an audio channel with given rate and sample data

Parameters

rate	sample rate in Hertz (1/s)
data	audio data as float vector

2.1.2.4 Channel::Channel (unsigned *rate*, unsigned *size*)

Create an audio channel with given rate and number of samples

Parameters

rate	sample rate in Hetz (1/s)
size	number of samples

2.1.3 Member Function Documentation

2.1.3.1 Channel Channel::downsample (unsigned *factor*) const

Downsample the channel by given factor.

Parameters

factor	downsample factor

Returns

new channel with a new sample frequency divided by the factor

2.1.3.2 Channel Channel::downsampleEnergy (unsigned *factor*) const

Downsample the channel by given factor and square the values

Parameters

factor	downsample factor

Returns

new channel with sample frequency divided by the factor

2.1.3.3 double Channel::l2downnorm (float *limit*) const

Returns the "down" l2-norm of the channel normalized to one sample, taking only the values into account that are higher than the given limit L. This hints to the energy of the channels when it is not active, when invoked with the l2norm of the channel as limit.

$$||u||_2^{<L} := \sqrt{\frac{1}{m}\sum_{j=0}^{m-1} v_j^2}, \text{ where } \forall i : u_i < L \exists_1 j(i) : v_{j(i)} = u_i$$

Parameters

limit	value

Returns

normalized "up" l2-norm of the channel

2.1.3.4 double Channel::l2norm (void) const

Returns the l2-norm of the channel normalized to one sample.

$$||u||_2 := \sqrt{\frac{1}{n}\sum_{i=0}^{n-1} u_i^2}$$

Returns

normalized l2-norm of the channel

2.1.3.5 double Channel::l2upnorm (float *limit*) const

Returns the "up" l2-norm of the channel normalized to one sample, taking only the values into account that are higher than the given limit L. This hints to the energy of the channels when it is active, when invoked with the l2norm of the channel as limit.

$$||u||_2^{>L} := \sqrt{\frac{1}{m}\sum_{j=0}^{m-1} v_j^2}, \text{ where } \forall i : u_i > L \exists_1 j(i) : v_{j(i)} = u_i$$

Parameters

limit	value

Returns

normalized "up" l2-norm of the channel

2.1.3.6 double Channel::linfnorm (void) const

Maximum absolute sample value

Returns

maximum absolute sample value

2.1.3.7 float & Channel::operator[ ] (int *index*)

Access a sample for read/write access The bounds are checked on the index and an impostor is returned in case of out-of-bounds requests.

Parameters

index	of sample

Returns

float reference on sample

2.1.3.8 float Channel::operator[ ] (int *index*) const

Access to a sample with read only access The bounds are checked on the index and zero is returned in case of out-of-bounds requests.

Parameters

index	of sample

Returns

float value of sample

2.1.3.9 Channel Channel::resampleTo (unsigned *newRate*) const

Create a copy of this channel with given sample rate

Parameters

newRate	sample rate of target channel

Returns

channel with given sample rate

2.1.3.10 Channel Channel::resizeTo (unsigned *size*) const

Create a copy of this channel with given size

Parameters

size	new number of samples

Returns

channel with given number of samples

2.1.3.11 unsigned Channel::samplerate ( ) const

Sample rate of this channel

Returns

sample rate in Hertz (1/s)

2.1.3.12 unsigned Channel::size ( ) const

Number of samples in this channel

Returns

number of samples

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

- src/Channel.h
- src/Channel.cpp

2.2 CrosstalkFilter Class Reference

The CrosstalkFilter tries to identify time-delayed crosstalk of each channel in other channels by comparing integrals of l2power and mutes identified sections.

```
#include <CrosstalkFilter.h>
```

Public Member Functions

- CrosstalkFilter (Channels &aChannels, unsigned aDownsampleLevel, unsigned aWorkwindow, unsigned a↩ MinShift, unsigned aMaxShift, float aMuteStartRatio=1.2, float aMuteFullRatio=1.5)
- CrosstalkFilter (Channels &aChannels, unsigned aDownsampleLevel, double windowsec=0.1, double mindistance=1.5, double maxdistance=5.0, float aMuteStartRatio=1.2, float aMuteFullRatio=1.5)
- void analyze2 ()
- void analyze ()
- void save (std::string)
- void mute ()

2.2.1 Detailed Description

The CrosstalkFilter tries to identify time-delayed crosstalk of each channel in other channels by comparing integrals of l2power and mutes identified sections.

Main issue is that it does not recognize the channel with more quality. It tends to mute channels less with more open mics than channels with directed mics. This is unwanted behaviour.

Starting from this two channel example, where the second channel consists of crosstalk from the first channel and a short original input.

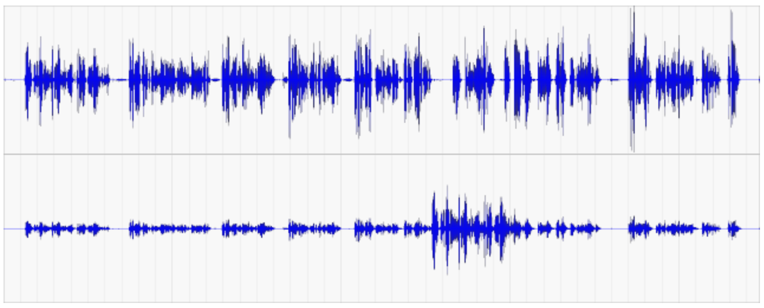


Figure 8 Exemplary two channel input

The crosstalk filter decreases the volume of the passages where mainly previous signals from other channels are detected.

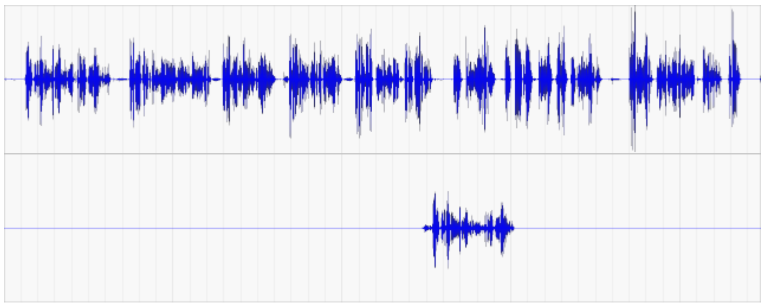


Figure 9 Result of crosstalk filter

2.2.2 Constructor & Destructor Documentation

2.2.2.1 CrosstalkFilter::CrosstalkFilter (Channels & *aChannels*, unsigned *aDownsampleLevel*, unsigned *aWorkwindow*, unsigned *aMinShift*, unsigned *aMaxShift*, float *aMuteStartRatio* = `1.2`, float *aMuteFullRatio* = `1.5`)

CrosstalkFilter Constructor with sample settings (please consider using the variant with physical settings!)

Parameters

aChannels	std::vector of channels the filter will operate on
aDownsampleLevel	factor of downsampling for the analysis
aWorkwindow	window size in samples the comparism is made on
aMinShift	number of minimum time-delay in samples (dependent on aDownsampleLevel)
aMaxShift	number of maximum time-delay in samples (dependent on aDownsampleLevel)
aMuteStartRatio	ratio of integrals of original to rest from where muting will linearly start
aMuteFullRatio	ratio of integrals of original to rest where linearity ends and full mute will occur

Returns

CrosstalkFilter object

See also

analyze() and mute()

2.2.2.2 CrosstalkFilter::CrosstalkFilter (Channels & *aChannels*, unsigned *aDownsampleLevel*, double *windowsec* = `0.1`, double *mindistance* = `1.5`, double *maxdistance* = `5.0`, float *aMuteStartRatio* = `1.2`, float *aMuteFullRatio* = `1.5`)

CrosstalkFilter Constructor with sample settings (please consider using the variant with physical settings!)

Parameters

aChannels	std::vector of channels the filter will operate on
aDownsampleLevel	factor of downsampling for the analysis
windowsec	window size in seconds the comparism is made on

Parameters

mindistance	minimum assumed spatial distance between channels (in meters)
maxdistance	maximum assumed spatial distance between channels (in meters)
aMuteStartRatio	ratio of integrals of original to rest from where muting will linearly start
aMuteFullRatio	ratio of integrals of original to rest where linearity ends and full mute will occur

Returns

CrosstalkFilter object

See also

analyze() and mute()

2.2.3 Member Function Documentation

2.2.3.1 void CrosstalkFilter::analyze ( )

CrosstalkFilter analysis Looks through all channels if sound bits of other channels are contained and sets mute vector correspondingly. Does not alter any channels.

2.2.3.2 void CrosstalkFilter::analyze2 ( )

CrosstalkFilter analysis Looks through all channels if sound bits of other channels are contained and sets mute vector correspondingly. Does not alter any channels.

2.2.3.3 void CrosstalkFilter::mute ( )

CrosstalkFilter mute channels applies mute filter to previously given channels

2.2.3.4 void CrosstalkFilter::save (std::string *name*)

CrosstalkFilter save mute channels saves muting filter as a wave file for analysis

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

- src/CrosstalkFilter.h
- src/CrosstalkFilter.cpp

2.3 CrosstalkGate Class Reference

Simple and robust crosstalk gate.

```
#include <CrosstalkGate.h>
```

Static Public Member Functions

- static void gate (Channels &aChannels, unsigned aDownsampleLevel, double windowsec=0.1, double mixsec=0.1)

2.3.1 Detailed Description

Simple and robust crosstalk gate.

Starting from this two channel example, where the second channel consists of crosstalk from the first channel and a short original input.

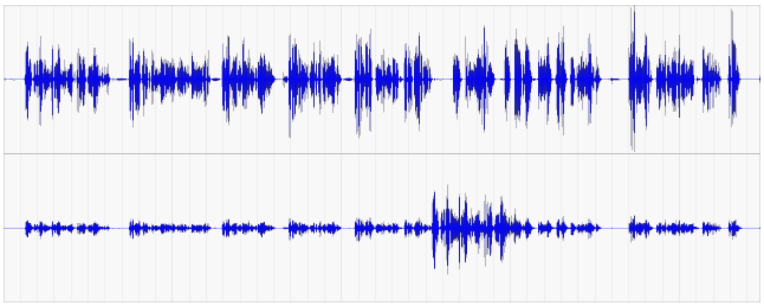


Figure 10 Exemplary two channel input

The crosstalk gate decreases the volume of the passages in the second channel where its channel activity is below its maximum.

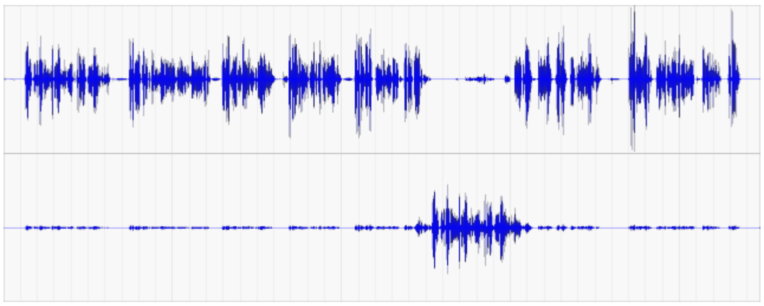


Figure 11 Result of crosstalk gate filter

2.3.2 Member Function Documentation

2.3.2.1 void CrosstalkGate::gate (Channels & *aChannels,* unsigned *aDownsampleLevel,* double *windowsec* = `0.1`, double *mixsec* = `0.1`) `[static]`

Crosstalk gate based on downsampled energy level of channels: For each channel the l2 energy of all sample below the averaged l2 level of the signal is taken as silence level. Then the activity in a window is assumed as factor above this silence level. (This should be limited in future.) The maximum activity will gain 100%, all other channels will be muted linearly due to their activity. The actual muting is averaged on a mixsec window to soften changes.

Parameters

aChannels	audio channels to work on
aDownsampleLevel	downsample factor
windowsec	activity window (in seconds)
mixsec	mixing average window (in seconds)

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

- src/CrosstalkGate.h
- src/CrosstalkGate.cpp

2.4 Equalizer Class Reference

Preset equalizer using frequency banding.

```
#include <Equalizer.h>
```

Static Public Member Functions

- static Channel bandedEqualizer (const Channel &c, std::vector< float > frequencies, std::vector< float > factors)
- static Channel voiceEnhance (const Channel &c)
- static Channels voiceEnhance (const Channels &c)

2.4.1 Detailed Description

Preset equalizer using frequency banding.

2.4.2 Member Function Documentation

2.4.2.1 Channel Equalizer::bandedEqualizer (const Channel & *c*, std::vector< float > *frequencies*, std::vector< float > *factors*) `[static]`

Amplification for seperate frequency bands

Parameters

c	audio channel to work on
frequencies	n cut-off frequencies
factors	n+1 amplication factors

Returns

resulting audio channel

2.4.2.2 Channel Equalizer::voiceEnhance (const Channel & *c*) `[static]`

Preset equalizer for voice channel

Parameters

c	audio channel to work on

Returns

resulting audio channel

2.4.2.3 Channels Equalizer::voiceEnhance (const Channels & *c*) `[static]`

Preset equalizer for voice channels

Parameters

c	audio channels to work on

Returns

resulting audio channel

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

- src/Equalizer.h
- src/Equalizer.cpp

2.5 Frequency Class Reference

Frequency filter class.

`#include <Frequency.h>`

Static Public Member Functions

- static Channels split (const Channel &a, float cutoff, float width=1000)
- static Channels split (Channel a, std::vector< float > cutoff, float width=1000)

2.5.1 Detailed Description

Frequency filter class.

2.5.2 Member Function Documentation

2.5.2.1 Channels Frequency::split (const Channel & *a*, float *cutoff*, float *width* = `1000`) `[static]`

Split the given channel in a high-frequency and low-frequency part

Parameters

a	given channel
cutoff	frequency
width	transition bandwidth

Returns

two channels with lower and higher frequency part

2.5.2.2 Channels Frequency::split (Channel *a*, std::vector< float > *cutoff*, float *width* = `1000`) `[static]`

Band filter a given channel with respect to cutoff frequencies

Parameters

a	given channel
cutoff	frequency vector (strictly ascending frequencies!)
width	transition bandwidth

Returns

band filtered channels

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

- src/Frequency.h
- src/Frequency.cpp

2.6 Log Class Reference

Logging class to specify output format and level. Use LOG(level) for logging.

```
#include <Log.h>
```

Static Public Member Functions

- static std::ostream & Get (std::string file, int line, TLogLevel level=logINFO)
- static void setOutput (std::ostream &o)
- static void setStandardOutput (std::ostream &o)
- static void setErrorOutput (std::ostream &o)
- static void setLoglevel (TLogLevel level)
- static void setShowFunction (bool show)
- static void setShowRuntime (bool show)
- static TLogLevel getLoglevel ()

2.6.1 Detailed Description

Logging class to specify output format and level. Use LOG(level) for logging.

2.6.2 Member Function Documentation

2.6.2.1 std::ostream & Log::Get (std::string *file*, int *line*, TLogLevel *level* = `logINFO`) `[static]`

Return suitable stream for logging level and write configured prefix

Parameters

file	Source file
line	Source line
level	Logging Level

Returns

output stream for logging

2.6.2.2 static TLogLevel Log::getLoglevel ( ) `[inline]`, `[static]`

request current logging level

Returns

current logging level

2.6.2.3 static void Log::setErrorOutput (std::ostream & *o*) `[inline]`, `[static]`

Set error output stream for logging

Parameters

o	output stream to use

2.6.2.4 static void Log::setLoglevel (TLogLevel *level*) `[inline]`, `[static]`

Set logging level to use

Parameters

level	logging level

2.6.2.5 static void Log::setOutput (std::ostream & *o*) `[inline]`, `[static]`

Set both error and standard logging output stream

Parameters

o	output stream to use

2.6.2.6 static void Log::setShowFunction (bool *show*) `[inline]`, `[static]`

Should source file be displayed while logging

Parameters

show	source file

2.6.2.7 static void Log::setShowRuntime (bool *show*) [inline], [static]

Should the running time be displayed while logging

Parameters

show	logging time

2.6.2.8 static void Log::setStandardOutput (std::ostream & *o*) [inline], [static]

Set standard output stream for logging

Parameters

o	output stream to use

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

- src/Log.h
- src/Log.cpp

2.7 Maximizer Class Reference

Amplification with constant factor and soft clipping by sigmoid function.

```
#include <Maximizer.h>
```

Static Public Member Functions

- static void amplify (Channel &channel, float factor, int order=4)
- static void amplify (Channels &channels, float factor, int order=4)
- static void amplifyDenoise (Channel &channel, float factor, float minlevel, int order=4)
- static void amplifyDenoise (Channels &channels, float factor, float minlevel, int order=4)
- static void normalize (Channel &channel, float level=32767.)
- static void normalize (Channels &channels, float level=32767.)

2.7.1 Detailed Description

Amplification with constant factor and soft clipping by sigmoid function.

This is the result of the factor filter:

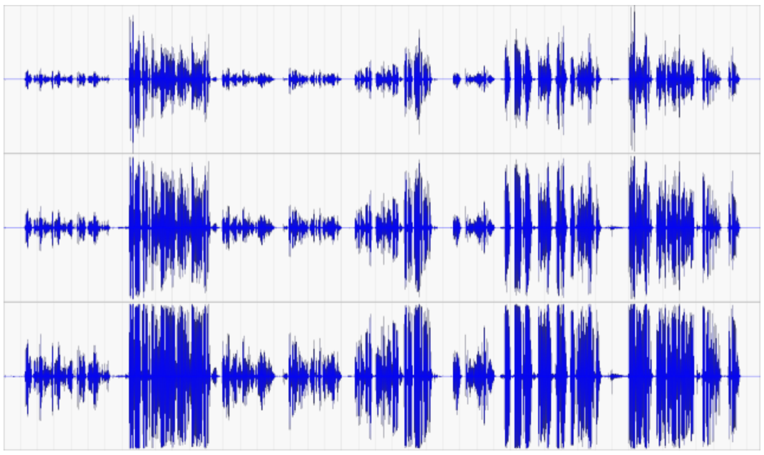


Figure 12 Result of factors 2 and 4

This is the result of the normalize filter:

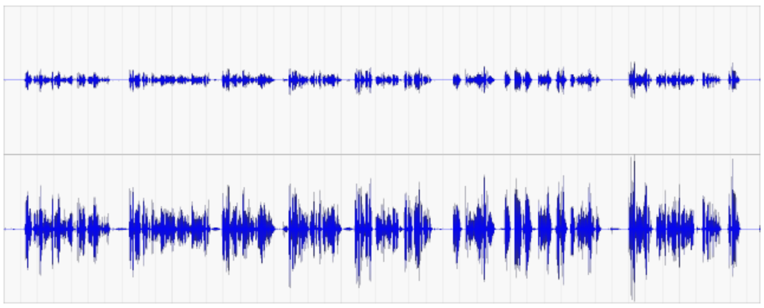


Figure 13 Result of normalization

2.7.2 Member Function Documentation

2.7.2.1 void Maximizer::amplify (Channel & *channel,* float *factor,* int *order* = 4) [static]

Multiplication of signal by constant factor and soft limiting by sigmoid function:

$$v(x) := \frac{c \cdot u(x)}{\sqrt[n]{1 + \left|\frac{c \cdot u(x)}{32000}\right|^n}}$$

Parameters

channel	audio segment to be multiplied
factor	factor
order	of sigmoid function

2.7.2.2 void Maximizer::amplify (Channels & *channels,* float *factor,* int *order =* `4`) `[static]`

Multiplication of signal by constant factor and soft limiting by sigmoid function:

$$v(x) := \frac{c \cdot u(x)}{\sqrt[n]{1 + \left|\frac{c \cdot u(x)}{32000}\right|^n}}$$

Parameters

channels	audio segments to be multiplied
factor	factor
order	of sigmoid function

2.7.2.3 void Maximizer::amplifyDenoise (Channel & *channel,* float *factor,* float *minlevel,* int *order =* `4`) `[static]`

Multiplication of signal by constant factor and soft limiting by sigmoid multiplied with quadratic root function:

$$v(x) := \frac{c \cdot u(x)}{\sqrt[n]{1 + \left|\frac{c \cdot u(x)}{32000}\right|^n}} \cdot \frac{(u(x))^2}{(u(x))^2 + \varrho^2}$$

Parameters

channel	audio segment to be multiplied
factor	factor
minlevel	noise voltage level
order	of sigmoid function

2.7.2.4 void Maximizer::amplifyDenoise (Channels & *channels,* float *factor,* float *minlevel,* int *order =* `4`) `[static]`

Multiplication of signal by constant factor and soft limiting by sigmoid multiplied with quadratic root function:

$$v(x) := \frac{c \cdot u(x)}{\sqrt[n]{1 + \left|\frac{c \cdot u(x)}{32000}\right|^n}} \cdot \frac{(u(x))^2}{(u(x))^2 + \varrho^2}$$

Parameters

channels	audio segments to be multiplied
factor	factor
minlevel	noise voltage level
order	of sigmoid function

2.7.2.5 void Maximizer::normalize (Channel & *channel,* float *level =* `32767.`) `[static]`

Normalize the maximum absolute value to given level.

Parameters

channel	audio segment to be normalized
level	target voltage level

2.7.2.6 void Maximizer::normalize (Channels & *channels*, float *level* = `32767.`) `[static]`

Normalize the maximum absolute value to given level.

Parameters

channels	audio segments to be normalized
level	target voltage level

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

- src/Maximizer.h
- src/Maximizer.cpp

2.8 Merge Class Reference

Merging of sound data segments (overlapping or fading)

```
#include <Merge.h>
```

Static Public Member Functions

- static Channels overlap (Channels &a, Channels &b, float sec)
- static Channels fade (Channels &a, Channels &b, float sec)

2.8.1 Detailed Description

Merging of sound data segments (overlapping or fading)

2.8.2 Member Function Documentation

2.8.2.1 Channels Merge::fade (Channels & *a*, Channels & *b*, float *sec*) `[static]`

Render the fading (i.e. with de- and increasing volume) of two sound data segments

Parameters

a	prior sound data segment
b	later sound data segment
sec	seconds of fading

Returns

resulting faded sound data segment

2.8.2.2 Channels Merge::overlap (Channels & *a*, Channels & *b*, float *sec*) [static]

Render the overlap (i.e. both on full volume) of two sound data segments

Parameters

a	prior sound data segment
b	later sound data segment
sec	seconds of overlap

Returns

resulting overlapped sound data segment

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

- src/Merge.h
- src/Merge.cpp

2.9 MonoMix Class Reference

Create mono mix-down.

`#include <MonoMix.h>`

Public Member Functions

- void mix (Channel &c, float factor=1.0)
- void mix (Channels &c)
- Channels & getTarget ()

2.9.1 Detailed Description

Create mono mix-down.

2.9.2 Member Function Documentation

2.9.2.1 Channels& MonoMix::getTarget ( ) [inline]

Return current mono mix-down

Returns

mix-down Channel

2.9.2.2 void MonoMix::mix (Channel & *c*, float *factor* = 1.0)

Mix one channel into the mix-down

Parameters

c	Channel
factor	intensity of rendering

2.9.2.3 void MonoMix::mix (**Channels & *c***)

Mix several channels into the mix-down

Parameters

c	Channels

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

- src/MonoMix.h
- src/MonoMix.cpp

2.10 OspacMain Class Reference

Main program class for dealing with command line options.

```
#include <OspacMain.h>
```

Public Member Functions

- OspacMain (std::vector< std::string >)
- int run (void)

Protected Types

- enum MixMode {
  SPATIAL, **STEREO**, **MONO**, **MULTI**,
  MaxMixMode }
- enum ArgMode { **VOICE**, **MIX**, **RAW**, **MaxArgMode** }
- enum TransitionMode { **NONE**, **OVERLAP**, **FADE**, **MaxTransitionMode** }

Protected Member Functions

- bool isOption (std::string &s)
- void setStandard ()
- void render (Channels &work, Channels &operand, Channels &target)

Protected Attributes

- std::vector< std::string > arg
- MixMode mixMode
- ArgMode argMode
- TransitionMode transitionMode
- float transitionSeconds
- TransitionMode nextTransitionMode
- float nextTransitionSeconds
- float maximizer
- bool normalizer
- bool leveler
- float levelTarget
- bool xGate
- bool xFilter
- bool skip
- float skipSilence
- bool voiceEq
- float stdMaximizer [MaxArgMode]
- bool stdNormalizer [MaxArgMode]
- bool stdLeveler [MaxArgMode]
- float stdLevelTarget [MaxArgMode]
- bool stdXGate [MaxArgMode]
- bool stdXFilter [MaxArgMode]
- bool stdSkip [MaxArgMode]
- float stdSkipSilence [MaxArgMode]
- bool stdVoiceEq [MaxArgMode]

Static Protected Attributes

- static std::string options [ ]

2.10.1 Detailed Description

Main program class for dealing with command line options.

This class represents the command line interface of ospac, and defines standard values for its settings.

2.10.2 Member Enumeration Documentation

2.10.2.1 enum **OspacMain::ArgMode** `[protected]`

Argument mode what kind of acoustic data is to be processed.

2.10.2.2 enum **OspacMain::MixMode** `[protected]`

Downmix mode for voice channels.

2.10.2.3 enum **OspacMain::TransitionMode** `[protected]`

Transition mode between acoustic data segments

2.10.3 Constructor & Destructor Documentation

2.10.3.1 OspacMain::OspacMain (std::vector< std::string > *aArg*)

Set up data and prepare everything for the run method.

Parameters

aArg	vector of command line options.

2.10.4 Member Function Documentation

2.10.4.1 bool OspacMain::isOption (std::string & *s*) `[protected]`

Tests the given parameter if it is an options by checking with options list

Parameters

s	parameter string to check

Returns

true if an option was detected

2.10.4.2 void OspacMain::render (Channels & *work,* Channels & *operand,* Channels & *target*) `[protected]`

Render last segment and current segment to the target segment according to all settings regarding current segment and transition

Parameters

work	previous audio data segment
operand	current audio data segment
target	target audio data segment

2.10.4.3 int OspacMain::run (void)

Run the application and do all actions that were requested my the option given.

Returns

0 in case of success, 1 in case of error.

2.10.4.4 void OspacMain::setStandard ( ) `[protected]`

Set all variables to their standard setting dependent on data mode

2.10.5 Member Data Documentation

2.10.5.1 std::vector<std::string> OspacMain::arg `[protected]`

Vector of all command line arguments

2.10.5.2 ArgMode OspacMain::argMode [protected]

Kind of acoustic data in this acoustic data segment

2.10.5.3 bool OspacMain::leveler [protected]

Should the current segment be levelled

2.10.5.4 float OspacMain::levelTarget [protected]

Leveling target energy

2.10.5.5 float OspacMain::maximizer [protected]

Current factor for maximizer

2.10.5.6 MixMode OspacMain::mixMode [protected]

Downmix mode for current acoustic data segment

2.10.5.7 TransitionMode OspacMain::nextTransitionMode [protected]

Transition mode to next acoustic data segment

2.10.5.8 float OspacMain::nextTransitionSeconds [protected]

Transition time to next acoustic data segment

2.10.5.9 bool OspacMain::normalizer [protected]

Should current segment be normalized

2.10.5.10 std::string OspacMain::options [static], [protected]

Initial value:

```
={"spatial","stereo","mono","multi",
                              "voice","mix","raw",
                              "fade","overlap",
                              "factor", "no-factor",
                              "leveler","no-leveler","target",
                              "normalize","no-normalize",
                              "skip","no-skip","skip-level",
                              "xgate","no-xgate",
                              "xfilter","no-xfilter",
                              "eqvoice","no-eqvoice",
                              "output",
                              "help","verbosity","plot"}
```

List of all valid command line tokens

2.10.5.11 bool OspacMain::skip [protected]

Should the current segment appy the skip-filter

2.10.5.12 float OspacMain::skipSilence [protected]

Silence detection for skipping filter

2.10.5.13 bool OspacMain::stdLeveler[MaxArgMode] [protected]

Standard leveler flag

2.10.5.14 float OspacMain::stdLevelTarget[MaxArgMode] [protected]

Standard leveler target

2.10.5.15 float OspacMain::stdMaximizer[MaxArgMode] [protected]

Standard maximizer factor

2.10.5.16 bool OspacMain::stdNormalizer[MaxArgMode] [protected]

Standard normalizer flag

2.10.5.17 bool OspacMain::stdSkip[MaxArgMode] [protected]

Standard skip filter flag

2.10.5.18 float OspacMain::stdSkipSilence[MaxArgMode] [protected]

Standard silence level for skip filter

2.10.5.19 bool OspacMain::stdVoiceEq[MaxArgMode] [protected]

Standard voice eq setting

2.10.5.20 bool OspacMain::stdXFilter[MaxArgMode] [protected]

Standard cross filter flat

2.10.5.21 bool OspacMain::stdXGate[MaxArgMode] [protected]

Standard cross gate flag

2.10.5.22 TransitionMode OspacMain::transitionMode [protected]

Transition mode from last acoustic data segment

2.10.5.23 float OspacMain::transitionSeconds [protected]

Transition time from last acoustic data segment

2.10.5.24 bool OspacMain::voiceEq [protected]

Should voice equalizer run over the channels?

2.10.5.25 bool OspacMain::xFilter [protected]

Should the current segment be filtered by the experimental cross-filter

2.10.5.26 bool OspacMain::xGate [protected]

Should the current segment be cross-gated

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

- src/OspacMain.h
- src/OspacMain.cpp

2.11 Physics Class Reference

Conversion of physical quantities.

```
#include <Physics.h>
```

Static Public Member Functions

- static double secToMeter (double s)
- static double meterToSec (double m)

2.11.1 Detailed Description

Conversion of physical quantities.

2.11.2 Member Function Documentation

2.11.2.1 static double Physics::meterToSec (double *m*) [inline], [static]

Convert meter distance to sound seconds

Parameters

m	meter distance

Returns

sound seconds

2.11.2.2 static double Physics::secToMeter (double *s*) [inline], [static]

Convert sound seconds to meter distance

Parameters

s	seconds

Returns

meter distance

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

- src/Physics.h

2.12 Plot Class Reference

Simple plots of audio channels.

```
#include <Plot.h>
```

Static Public Member Functions

- static void createPGMPlot (const Channels &channels, std::string name, unsigned sizeX=1280, unsigned sizeY=251)
- static void createPGMPlot (const Channel &channel, std::string name, unsigned sizeX=1280, unsigned sizeY=251)
- static void createPPMPlot (const Channels &channels, std::string name, unsigned sizeX=1280, unsigned sizeY=251)
- static void createPPMPlot (const Channel &channel, std::string name, unsigned sizeX=1280, unsigned sizeY=251)

2.12.1 Detailed Description

Simple plots of audio channels.

2.12.2 Member Function Documentation

2.12.2.1 void Plot::createPGMPlot (const Channels & *channels*, std::string *name*, unsigned *sizeX* = 1280, unsigned *sizeY* = 251) [static]

Create PGM plot of audio channels

Parameters

channels	the channels to plot
name	target file name
sizeX	width
sizeY	height per channel

2.12.2.2 void Plot::createPGMPlot (const Channel & *channel*, std::string *name*, unsigned *sizeX* = `1280`, unsigned *sizeY* = `251`) `[static]`

Create PGM plot of an audio channel

Parameters

channel	the channel to plot
name	target file name
sizeX	width
sizeY	height

2.12.2.3 void Plot::createPPMPlot (const Channels & *channels*, std::string *name*, unsigned *sizeX* = `1280`, unsigned *sizeY* = `251`) `[static]`

Create PPM plot of audio channels

Parameters

channels	the channels to plot
name	target file name
sizeX	width
sizeY	height per channel

2.12.2.4 void Plot::createPPMPlot (const Channel & *channel*, std::string *name*, unsigned *sizeX* = `1280`, unsigned *sizeY* = `251`) `[static]`

Create PPM plot of an audio channel

Parameters

channel	the channel to plot
name	target file name
sizeX	width
sizeY	height

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

- src/Plot.h
- src/Plot.cpp

2.13 SelectiveLeveler Class Reference

Selective Leveling by windowed average l2 energy Contains experimental code for constant leveling in tolerance area.

```
#include <SelectiveLeveler.h>
```

Static Public Member Functions

- static void level (Channels &aChannels, float targetL2, double windowSec, float minFraction, float silent↩ Fraction, float forwardWindowSec, float backWindowSec)
- static void level (Channel &aChannel, float targetL2, double windowSec, float minFraction, float silentFraction, float forwardWindowSec, float backWindowSec)
- static void levelStereo (Channels &aChannels, float targetL2, double windowSec, float minFraction, float silentFraction, float forwardWindowSec, float backWindowSec)
- static void levelStereo (Channel &aChannel, Channel &bChannel, float targetL2, double windowSec, float minFraction, float silentFraction, float forwardWindowSec, float backWindowSec)

2.13.1 Detailed Description

Selective Leveling by windowed average l2 energy Contains experimental code for constant leveling in tolerance area.

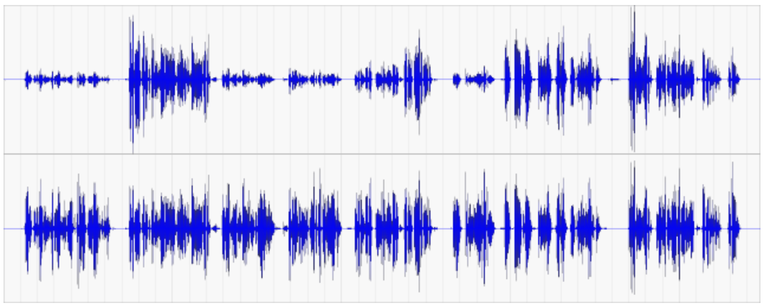


Figure 14 Result of selective leveler

2.13.2 Member Function Documentation

2.13.2.1 void SelectiveLeveler::level (Channels & *aChannels,* float *targetL2,* double *windowSec,* float *minFraction,* float *silentFraction,* float *forwardWindowSec,* float *backWindowSec*) [static]

Compute windowed average l2 energy. If the energy is below silent fraction, the signal is muted. If the energy is between silent fraction to minFraction compared to the maximal windows l2 energy it is linearily damped. The actual damping factor is windowed by forward and backwards window interval.

Parameters

aChannels	channels to do the individual leveling on
targetL2	target average l2 energy
windowSec	window size in seconds for l2 average energy
minFraction	fraction compared to l2 maximal value assumed signal
silentFraction	fraction compared to l2 maximal value assumed silence
forwardWindowSec	average forward part of window for factor application
backWindowSec	average backward part of window for factor application

2.13.2.2 void SelectiveLeveler::level (Channel & *aChannel,* float *targetL2,* double *windowSec,* float *minFraction,* float *silentFraction,* float *forwardWindowSec,* float *backWindowSec*) [static]

Compute windowed average l2 energy. If the energy is below silent fraction, the signal is muted. If the energy is between silent fraction to minFraction compared to the maximal windows l2 energy it is linearily damped. The actual damping factor is windowed by forward and backwards window interval.

Parameters

aChannel	channel to do the individual leveling on
targetL2	target average l2 energy
windowSec	window size in seconds for l2 average energy
minFraction	fraction compared to l2 maximal value assumed signal
silentFraction	fraction compared to l2 maximal value assumed silence
forwardWindowSec	average forward part of window for factor application
backWindowSec	average backward part of window for factor application

2.13.2.3 void SelectiveLeveler::levelStereo (Channels & *aChannels,* float *targetL2,* double *windowSec,* float *minFraction,* float *silentFraction,* float *forwardWindowSec,* float *backWindowSec*) [static]

Compute windowed average l2 energy. If the energy is below silent fraction, the signal is muted. If the energy is between silent fraction to minFraction compared to the maximal windows l2 energy it is linearily damped. The actual damping factor is windowed by forward and backwards window interval. This function each considers two channels for analysis.

Parameters

aChannels	channels to do the individual leveling on
targetL2	target average l2 energy
windowSec	window size in seconds for l2 average energy
minFraction	fraction compared to l2 maximal value assumed signal
silentFraction	fraction compared to l2 maximal value assumed silence
forwardWindowSec	average forward part of window for factor application
backWindowSec	average backward part of window for factor application

2.13.2.4 void SelectiveLeveler::levelStereo (Channel & *aChannel,* Channel & *bChannel,* float *targetL2,* double *windowSec,* float *minFraction,* float *silentFraction,* float *forwardWindowSec,* float *backWindowSec*) [static]

Compute windowed average l2 energy. If the energy is below silent fraction, the signal is muted. If the energy is between silent fraction to minFraction compared to the maximal windows l2 energy it is linearily damped. The actual damping factor is windowed by forward and backwards window interval. This function each considers two channels for analysis.

Parameters

aChannel	left channel to do the joint leveling on
bChannel	right channel to do the joint leveling on
targetL2	target average l2 energy
windowSec	window size in seconds for l2 average energy
minFraction	fraction compared to l2 maximal value assumed signal

Parameters

silentFraction	fraction compared to l2 maximal value assumed silence
forwardWindowSec	average forward part of window for factor application
backWindowSec	average backward part of window for factor application

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

- src/SelectiveLeveler.h
- src/SelectiveLeveler.cpp

2.14 Skip Class Reference

Skip silence.

```
#include <Skip.h>
```

Static Public Member Functions

- static float silence (Channels &channels, float silenceLevel=0.01, float minsec=0.5, float mintransition=0.05, float reductionOrder=0.75)

2.14.1 Detailed Description

Skip silence.

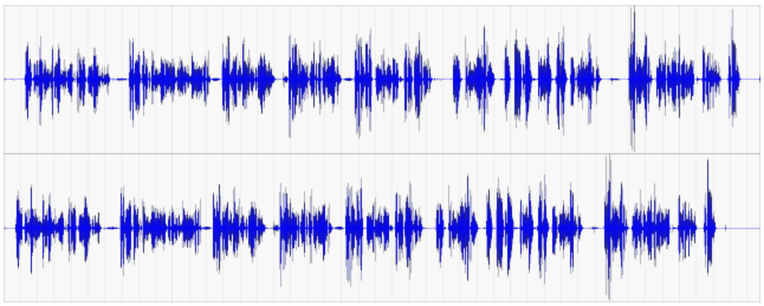


Figure 15 Result of standard skip filter

2.14.2 Member Function Documentation

2.14.2.1 float Skip::silence (Channels & *channels*, float *silenceLevel* = `0.01`, float *minsec* = `0.5`, float *mintransition* = `0.05`, float *reductionOrder* = `0.75`) `[static]`

Skip silence in channels if absolute sum of voltages are below silence level fraction compared to maximum level for longer than minsec seconds. Shorten the period by the time to the reduction order. The transition period is in the middle of silence for a maximum time of maxtransition seconds.

Parameters

channels	Channels where silence is to be skipped
silenceLevel	fraction compared to maximum what is considered silence
minsec	minimum time of silence before skipping is considered
mintransition	minimum time of transition
reductionOrder	reduction by time to the reduction order

Returns

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

- src/Skip.h
- src/Skip.cpp

2.15 StereoMix Class Reference

Create stereo mixdown of channels.

`#include <StereoMix.h>`

Public Member Functions

- StereoMix ()
- void mix (Channel &c, float leftFactor, float rightFactor, float leftDistance, float rightDistance)
- void mixBanded (Channel &c, float leftFactor, float rightFactor, float leftDistance, float rightDistance)
- void mix (Channels &c, bool spatial=false, bool banded=false)
- Channels & getTarget ()

2.15.1 Detailed Description

Create stereo mixdown of channels.

2.15.2 Constructor & Destructor Documentation

2.15.2.1 StereoMix::StereoMix ( )

Create initial target

2.15.3 Member Function Documentation

2.15.3.1 Channels& StereoMix::getTarget ( ) `[inline]`

Request current stereo mixdown

Returns

stereo mixdown

2.15.3.2 void StereoMix::mix (Channel & *c*, float *leftFactor*, float *rightFactor*, float *leftDistance*, float *rightDistance*)

Mix single Channel into the target

Parameters

c	Channel
leftFactor	Rendering intensity left target channel
rightFactor	Rendering intensity right target channel
leftDistance	Distance in meter from left channel
rightDistance	Distance in meter from right channel

2.15.3.3 void StereoMix::mix (Channels & *c*, bool *spatial* = `false`, bool *banded* = `false`)

Mix channels into target using equidistant positions

Parameters

c	Channels
spatial	Use spatial delay?
banded	Use frequency dependence?

2.15.3.4 void StereoMix::mixBanded (Channel & *c*, float *leftFactor*, float *rightFactor*, float *leftDistance*, float *rightDistance*)

Mix single Channel into target with frequency dependence

Parameters

c	Channel
leftFactor	Rendering intensity left target channel
rightFactor	Rendering intensity right target channel
leftDistance	Distance in meter from left channel
rightDistance	Distance in meter from right channel

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

- src/StereoMix.h
- src/StereoMix.cpp

2.16 Wave Class Reference

Wave-file loading and saving via libsndfile.

```
#include <Wave.h>
```

Static Public Member Functions

- static Channels load (const std::string &)
- static Channels & load (const std::string &, Channels &target)
- static int save (const std::string &, Channels &)
- static int save (const std::string &, Channel &)

2.16.1 Detailed Description

Wave-file loading and saving via libsndfile.

2.16.2 Member Function Documentation

2.16.2.1 Channels Wave::load (const std::string & *name*) [static]

Load a wave file from the file system using libsndfile.

Parameters

name	file system name of file

Returns

Channels containing the wave channels

2.16.2.2 Channels & Wave::load (const std::string & *name*, Channels & *target*) [static]

Load a wave file from the file system using libsndfile, avoiding copy operations.

Parameters

name	file system name of file
target	Channel object to save the data in

Returns

Channels references containing the wave channels

2.16.2.3 int Wave::save (const std::string & *name*, Channels & *channels*) [static]

Save a multi-channel wave file to the file system using libsndfile. The sample data is assumed to be in the range of [-32767,32767] and entries beyond are limited to the range.

Parameters

name	file system name of file
channels	channels to be saved.

Returns

0 in case of success, 1 in case of error.

2.16.2.4 int Wave::save (const std::string & *name*, Channel & *channel*) [static]

Save a single-channel wave file to the file system using libsndfile. The sample data is assumed to be in the range of [-32767,32767] and entries beyond are limited to the range.

Parameters

name	file system name of file
channel	channels to be saved.

Returns

0 in case of success, 1 in case of error.

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

- src/Wave.h
- src/Wave.cpp

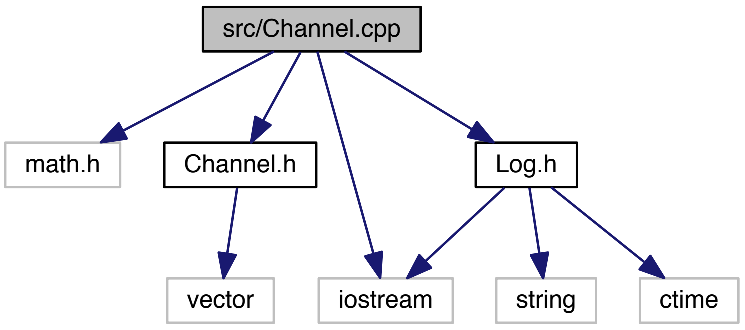
3 File Documentation

3.1 src/Channel.cpp File Reference

Audio channel abstraction.

```
#include <math.h>
#include <iostream>
#include "Channel.h"
#include "Log.h"
```

Include dependency graph for Channel.cpp:



3.1.1 Detailed Description

Audio channel abstraction.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

5.2.2016

Copyright

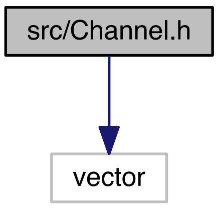
MIT License (see LICENSE file)

3.2 src/Channel.h File Reference

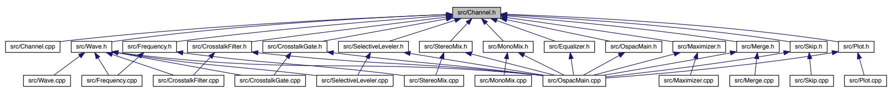
Audio channel abstraction.

```
#include <vector>
```

Include dependency graph for Channel.h:



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



Classes

- class Channel

  Audio channel abstraction class.

Typedefs

- typedef std::vector< Channel > Channels

3.2.1 Detailed Description

Audio channel abstraction.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

5.2.2016

Copyright

MIT License (see LICENSE file)

3.2.2 Typedef Documentation

3.2.2.1 typedef std::vector<Channel> Channels

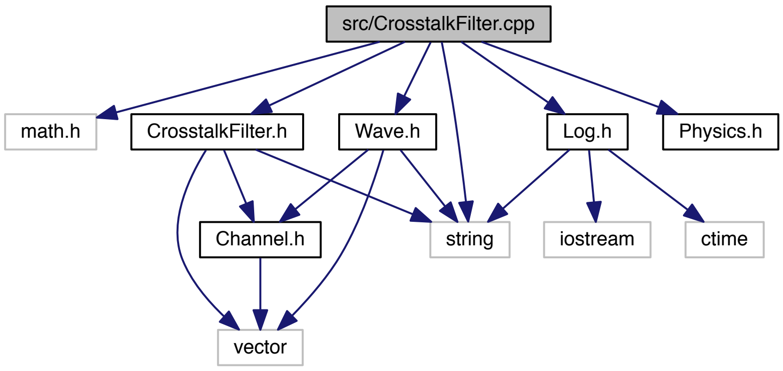
Vector of channels as type for multiple channels.

3.3 src/CrosstalkFilter.cpp File Reference

Filter to actively identify crosstalk in other channels.

```
#include <math.h>
#include <string>
#include "CrosstalkFilter.h"
#include "Wave.h"
#include "Physics.h"
#include "Log.h"
```

Include dependency graph for CrosstalkFilter.cpp:



3.3.1 Detailed Description

Filter to actively identify crosstalk in other channels.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

6.2.2016

Copyright

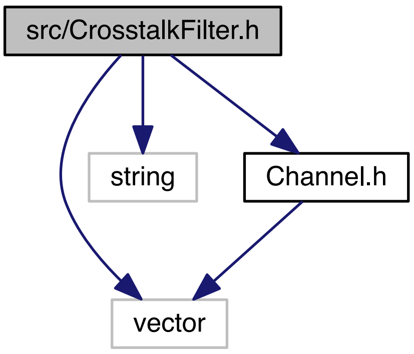
MIT License (see LICENSE file)

3.4 src/CrosstalkFilter.h File Reference

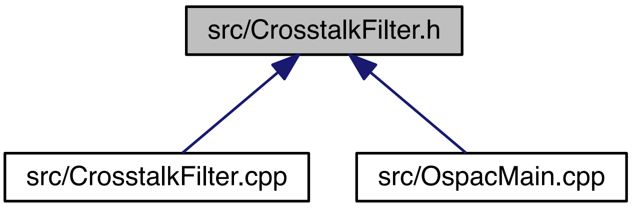
Filter to actively identify crosstalk in other channels.

```
#include <vector>
#include <string>
#include "Channel.h"
```

Include dependency graph for CrosstalkFilter.h:



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



Classes

- class CrosstalkFilter

  The CrosstalkFilter tries to identify time-delayed crosstalk of each channel in other channels by comparing integrals of l2power and mutes identified sections.

3.4.1 Detailed Description

Filter to actively identify crosstalk in other channels.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

6.2.2016

Copyright

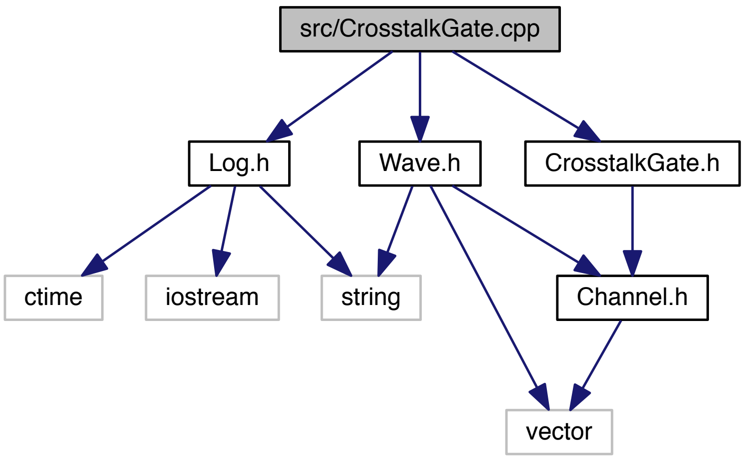
MIT License (see LICENSE file)

3.5 src/CrosstalkGate.cpp File Reference

Crosstalk gate mutes less active channels.

```
#include "CrosstalkGate.h"
#include "Log.h"
#include "Wave.h"
```

Include dependency graph for CrosstalkGate.cpp:



3.5.1 Detailed Description

Crosstalk gate mutes less active channels.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

9.2.2016

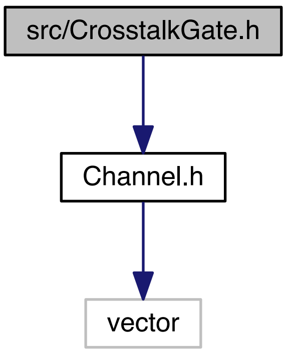
Copyright

MIT License (see LICENSE file)

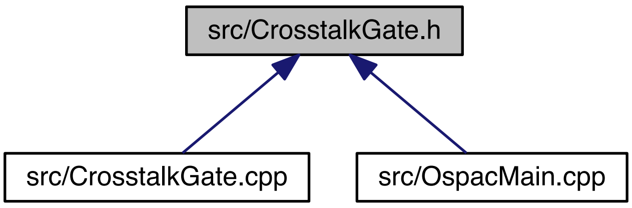
3.6 src/CrosstalkGate.h File Reference

Crosstalk gate mutes less active channels.

`#include "Channel.h"`
Include dependency graph for CrosstalkGate.h:



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



Classes

- class CrosstalkGate

  Simple and robust crosstalk gate.

3.6.1 Detailed Description

Crosstalk gate mutes less active channels.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

9.2.2016

Copyright

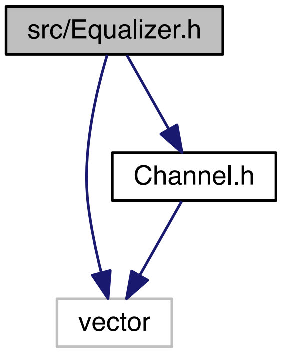
MIT License (see LICENSE file)

3.7 src/Equalizer.h File Reference

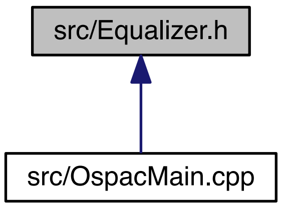
Equalizer for sound enhancement.

```
#include <vector>
#include "Channel.h"
```

Include dependency graph for Equalizer.h:



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



Classes

- class Equalizer

  Preset equalizer using frequency banding.

3.7.1 Detailed Description

Equalizer for sound enhancement.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

3.3.2016

Copyright

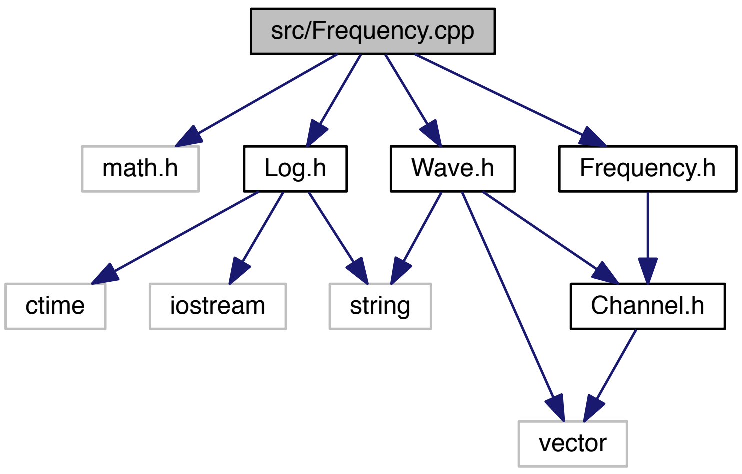
MIT License (see LICENSE file)

3.8 src/Frequency.cpp File Reference

Frequency filters.

```
#include <math.h>
#include "Frequency.h"
#include "Log.h"
#include "Wave.h"
```

Include dependency graph for Frequency.cpp:



3.8.1 Detailed Description

Frequency filters.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

14.2.2016

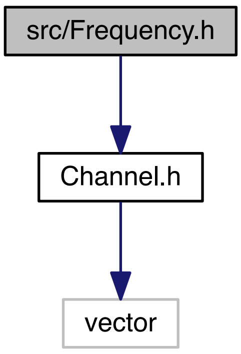
Copyright

MIT License (see LICENSE file)

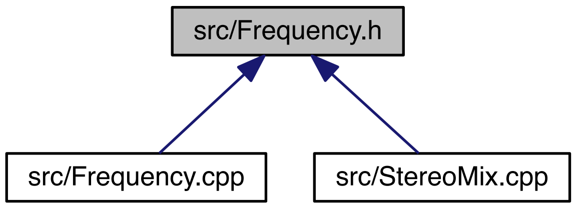
3.9 src/Frequency.h File Reference

Frequency filters.

`#include "Channel.h"`
Include dependency graph for Frequency.h:



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



Classes

- class Frequency

  Frequency filter class.

3.9.1 Detailed Description

Frequency filters.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

14.2.2016

Copyright

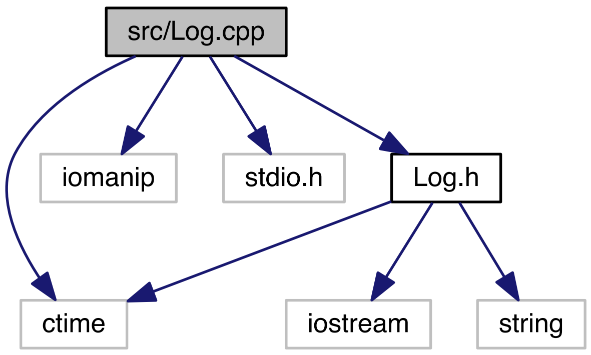
MIT License (see LICENSE file)

3.10 src/Log.cpp File Reference

Efficient logging stream.

```
#include <ctime>
#include <iomanip>
#include <stdio.h>
#include "Log.h"
```

Include dependency graph for Log.cpp:



3.10.1 Detailed Description

Efficient logging stream.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

7.2.2016

Copyright

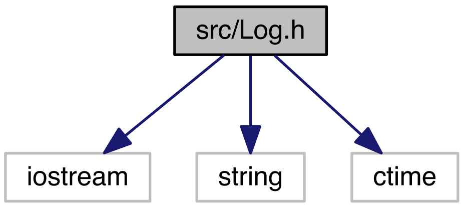
MIT License (see LICENSE file) Inspired by http://stackoverflow.com/questions/524524/creating-an-o and http://www.drdobbs.com/article/print?articleId=201804215&siteSection↩
Name=cpp

3.11 src/Log.h File Reference

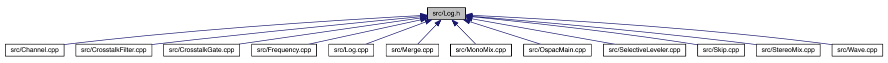
Efficient logging stream.

```
#include <iostream>
#include <string>
#include <ctime>
```

Include dependency graph for Log.h:



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



Classes

- class Log

  Logging class to specify output format and level. Use LOG(level) for logging.

Macros

- #define LOG(level)

  A macro for efficient creation of logging stream.

Enumerations

- enum TLogLevel {
  logFATAL, **logERROR**, **logWARNING**, **logINFO**,
  logDEBUG, **logDEBUG1**, **logDEBUG2**, **logDEBUG3**,
  logDEBUG4 }

3.11.1 Detailed Description

Efficient logging stream.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

7.2.2016

Copyright

MIT License (see LICENSE file) Inspired by http://stackoverflow.com/questions/524524/creating-an-os and http://www.drdobbs.com/article/print?articleId=201804215&siteSection↩ Name=cpp

3.11.2 Macro Definition Documentation

3.11.2.1 #define LOG(*level*)

Value:

```
if (level > Log::getLoglevel()) ; \
else Log::Get(__FILE__,__LINE__,level)
```

A macro for efficient creation of logging stream.

Use this macro for logging information that is only computed when logging is activiated at given level: It expands to an if-statement that computes the right hand side only if the logging level is reached.

Example: LOG(logDEBUG) << "test information " << expensiveFunkction << std::endl; expands to if(logDEBUG > LOG::getLoglevel()) ; // do nothing else LOG::GET(**FILE**,**LINE**,logDEBUG) << "test information " ...

3.11.3 Enumeration Type Documentation

3.11.3.1 enum **TLogLevel**

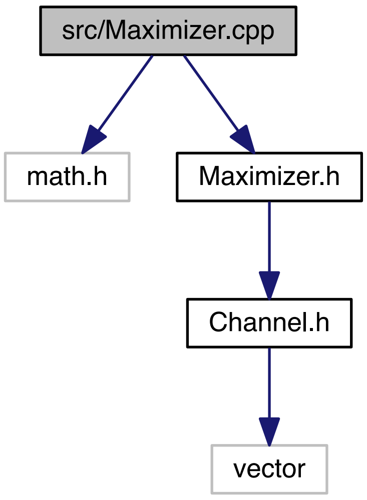
Logging Levels

3.12 src/Maximizer.cpp File Reference

Amplification and normalization.

```
#include <math.h>
#include "Maximizer.h"
```

Include dependency graph for Maximizer.cpp:



3.12.1 Detailed Description

Amplification and normalization.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

7.2.2016

Copyright

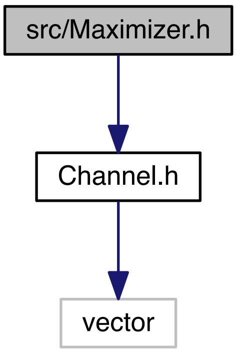
MIT License (see LICENSE file)

3.13 src/Maximizer.h File Reference

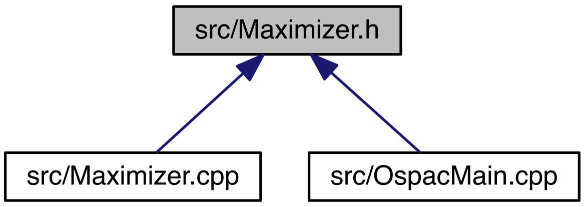
Amplification and normalization.

```
#include "Channel.h"
```

Include dependency graph for Maximizer.h:



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



Classes

- class Maximizer

    Amplification with constant factor and soft clipping by sigmoid function.

3.13.1 Detailed Description

Amplification and normalization.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

7.2.2016

Copyright

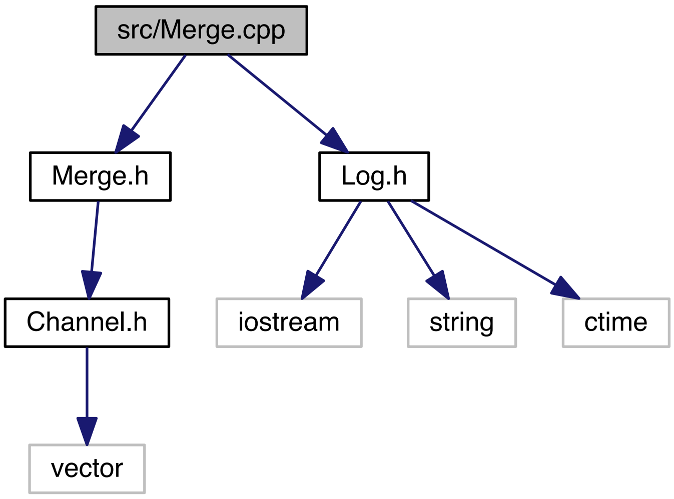
MIT License (see LICENSE file)

3.14 src/Merge.cpp File Reference

Merging of audio segments either with overlap or fading.

```
#include "Merge.h"
#include "Log.h"
```

Include dependency graph for Merge.cpp:



3.14.1 Detailed Description

Merging of audio segments either with overlap or fading.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

11.2.2016

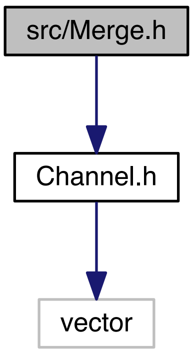
Copyright

MIT License (see LICENSE file)

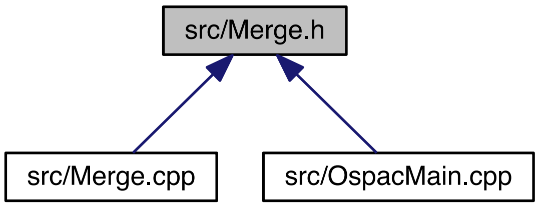
3.15 src/Merge.h File Reference

Merging of audio segments either with overlap or fading.

`#include "Channel.h"`
Include dependency graph for Merge.h:



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



Classes

- class Merge

  Merging of sound data segments (overlapping or fading)

3.15.1 Detailed Description

Merging of audio segments either with overlap or fading.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

11.2.2016

Copyright

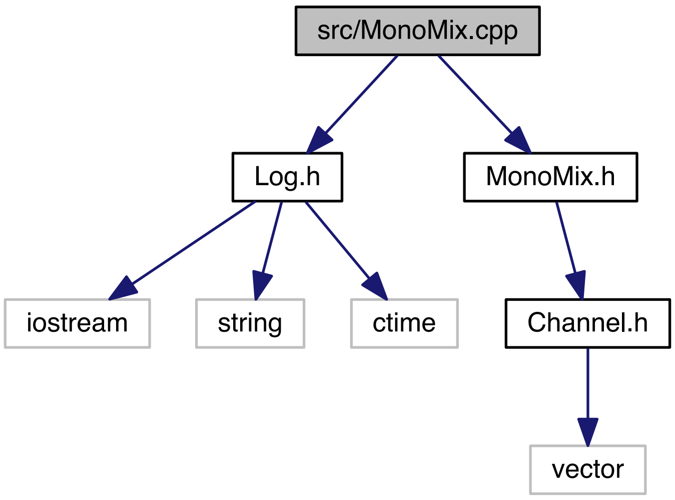
MIT License (see LICENSE file)

3.16 src/MonoMix.cpp File Reference

Mono mix-down.

```
#include "Log.h"
#include "MonoMix.h"
```

Include dependency graph for MonoMix.cpp:



3.16.1 Detailed Description

Mono mix-down.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

12.2.2016

Copyright

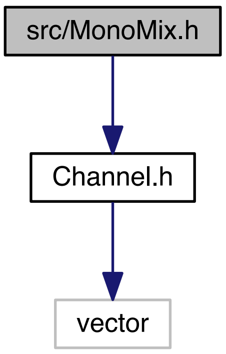
MIT License (see LICENSE file)

3.17 src/MonoMix.h File Reference

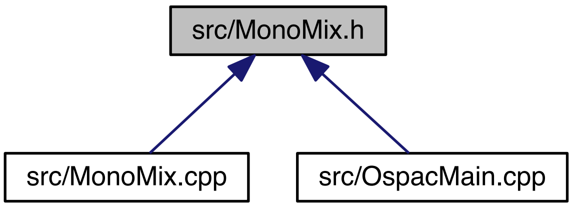
Mono mix-down.

```
#include "Channel.h"
```

Include dependency graph for MonoMix.h:



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



Classes

- class MonoMix

    Create mono mix-down.

3.17.1 Detailed Description

Mono mix-down.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

12.2.2016

Copyright

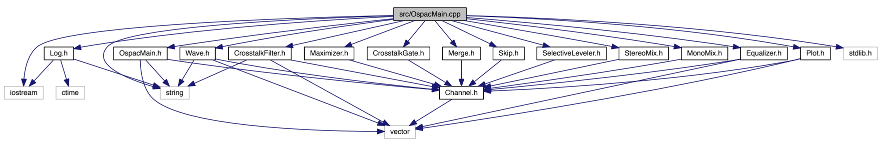
MIT License (see LICENSE file)

3.18 src/OspacMain.cpp File Reference

Main function and command line interface.

```
#include <iostream>
#include <string>
#include "OspacMain.h"
#include "Wave.h"
#include "CrosstalkFilter.h"
#include "Log.h"
#include "SelectiveLeveler.h"
#include "StereoMix.h"
#include "MonoMix.h"
#include "Maximizer.h"
#include "CrosstalkGate.h"
#include "Merge.h"
#include "Skip.h"
#include "Equalizer.h"
#include "Plot.h"
#include <stdlib.h>
```

Include dependency graph for OspacMain.cpp:



Functions

- int main (int argc, char ∗argv[ ])

3.18.1 Detailed Description

Main function and command line interface.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

5.2.2016

Copyright

MIT License (see LICENSE file)

3.18.2 Function Documentation

3.18.2.1 int main (int *argc,* char ∗ *argv[ ]*)

Main program entry point

Parameters

argc	number of arguments
argv	argument strings

Returns

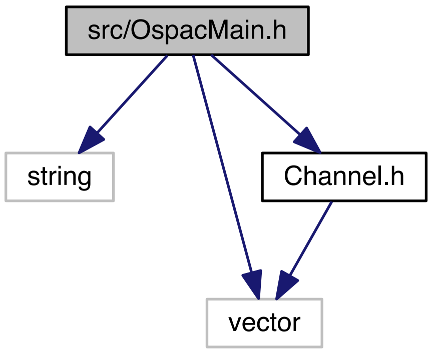
0 for success, all others for errors

3.19 src/OspacMain.h File Reference

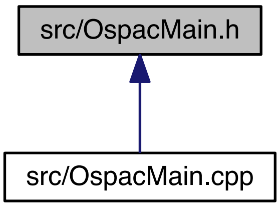
Command line interface.

```
#include <string>
#include <vector>
#include "Channel.h"
```

Include dependency graph for OspacMain.h:



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



Classes

- class OspacMain

  Main program class for dealing with command line options.

3.19.1 Detailed Description

Command line interface.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

5.2.2016

Copyright

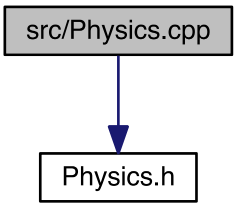
MIT License (see LICENSE file)

3.20 src/Physics.cpp File Reference

Conversion of physical quantities.

```
#include "Physics.h"
```

Include dependency graph for Physics.cpp:



3.20.1 Detailed Description

Conversion of physical quantities.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

7.2.2016

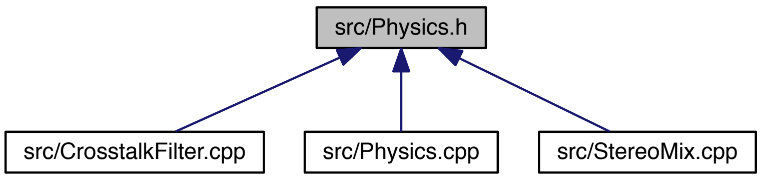
Copyright

MIT License (see LICENSE file)

3.21 src/Physics.h File Reference

Conversion of physical quantities.

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



Classes

- class Physics

  Conversion of physical quantities.

Variables

- const float v_Schall =343.2

3.21.1 Detailed Description

Conversion of physical quantities.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

7.2.2016

Copyright

MIT License (see LICENSE file)

3.21.2 Variable Documentation

3.21.2.1 const float v_Schall =343.2

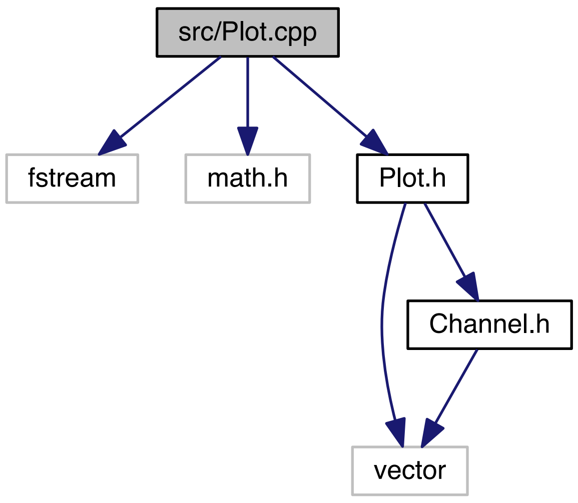
Speed of sound in air (at room temperature)

3.22 src/Plot.cpp File Reference

Simple plots of audio channels.

```
#include <fstream>
#include <math.h>
#include "Plot.h"
```

Include dependency graph for Plot.cpp:



3.22.1 Detailed Description

Simple plots of audio channels.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

6.3.2016

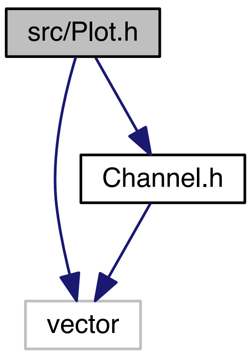
Copyright

MIT License (see LICENSE file)

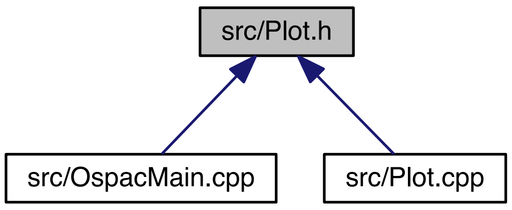
3.23 src/Plot.h File Reference

Simple plots of audio channels.

```
#include <vector>
#include "Channel.h"
```
Include dependency graph for Plot.h:



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



Classes

- class Plot

  Simple plots of audio channels.

3.23.1 Detailed Description

Simple plots of audio channels.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

6.3.2016

Copyright

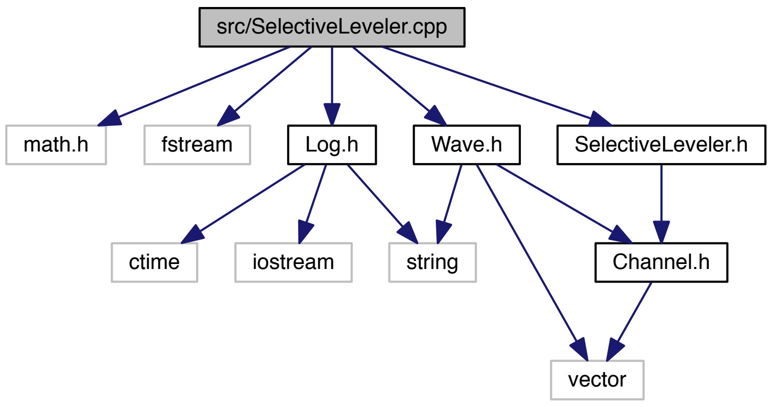
MIT License (see LICENSE file)

3.24 src/SelectiveLeveler.cpp File Reference

Selective Leveler working on windowed l2 energy of signal.

```
#include <math.h>
#include <fstream>
#include "SelectiveLeveler.h"
#include "Log.h"
#include "Wave.h"
```

Include dependency graph for SelectiveLeveler.cpp:



3.24.1 Detailed Description

Selective Leveler working on windowed l2 energy of signal.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

7.2.2016

Copyright

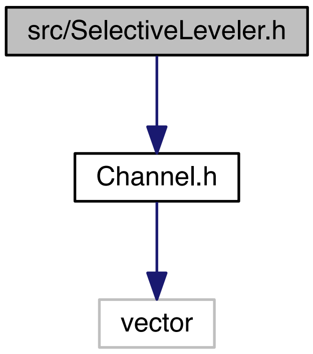
MIT License (see LICENSE file)

3.25 src/SelectiveLeveler.h File Reference

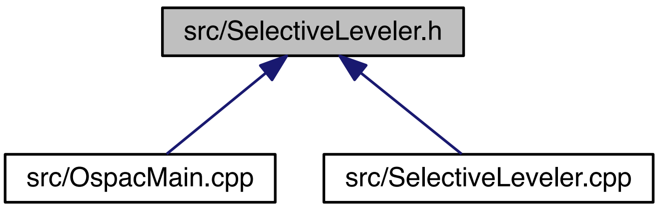
Selective Leveler working on windowed l2 energy of signal.

```
#include "Channel.h"
```

Include dependency graph for SelectiveLeveler.h:



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



Classes

- class SelectiveLeveler

  Selective Leveling by windowed average l2 energy Contains experimental code for constant leveling in tolerance area.

3.25.1 Detailed Description

Selective Leveler working on windowed l2 energy of signal.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

7.2.2016

Copyright

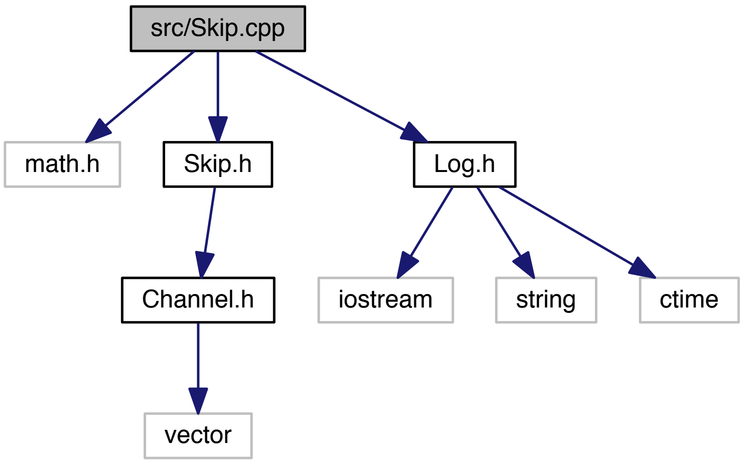
MIT License (see LICENSE file)

3.26 src/Skip.cpp File Reference

Skip silence.

```
#include <math.h>
#include "Skip.h"
#include "Log.h"
```

Include dependency graph for Skip.cpp:



3.26.1 Detailed Description

Skip silence.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

18.2.2016

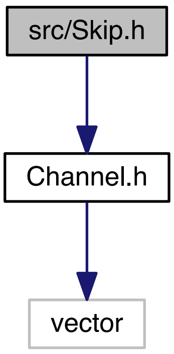
Copyright

MIT License (see LICENSE file)

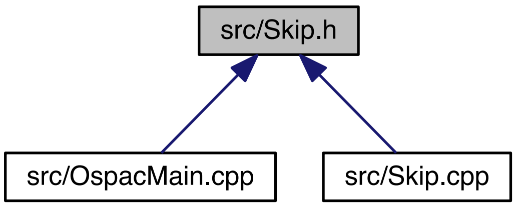
3.27 src/Skip.h File Reference

Skip silence.

`#include "Channel.h"`
Include dependency graph for Skip.h:



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



Classes

- class Skip

  Skip silence.

3.27.1 Detailed Description

Skip silence.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

18.2.2016

Copyright

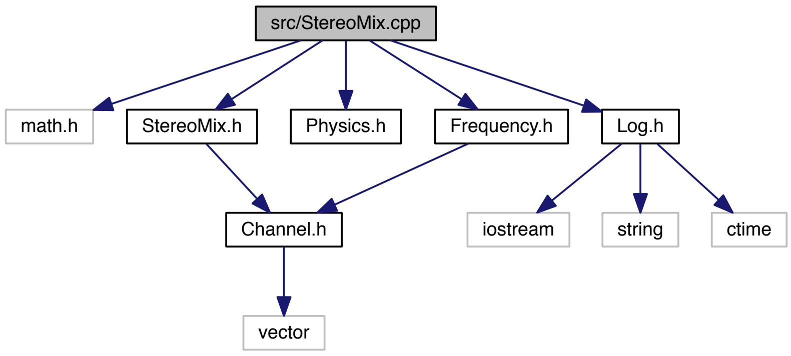
MIT License (see LICENSE file)

3.28 src/StereoMix.cpp File Reference

Stereo mix-down.

```
#include <math.h>
#include "StereoMix.h"
#include "Physics.h"
#include "Log.h"
#include "Frequency.h"
```

Include dependency graph for StereoMix.cpp:



3.28.1 Detailed Description

Stereo mix-down.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

7.2.2016

Copyright

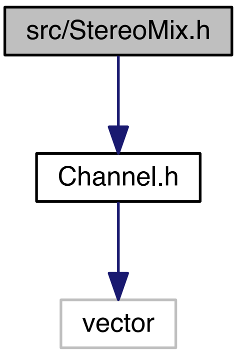
MIT License (see LICENSE file)

3.29 src/StereoMix.h File Reference

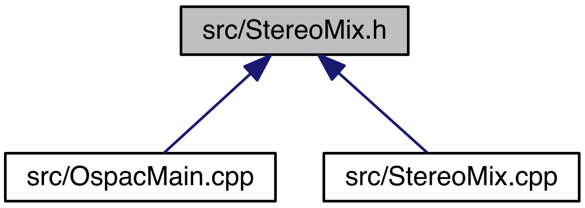
Stereo mix-down.

```
#include "Channel.h"
```

Include dependency graph for StereoMix.h:



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



Classes

- class StereoMix

    Create stereo mixdown of channels.

3.29.1 Detailed Description

Stereo mix-down.

Author

Sebastian Ritterbusch ospac@ritterbusch.de

Version

1.0

Date

7.2.2016

Copyright

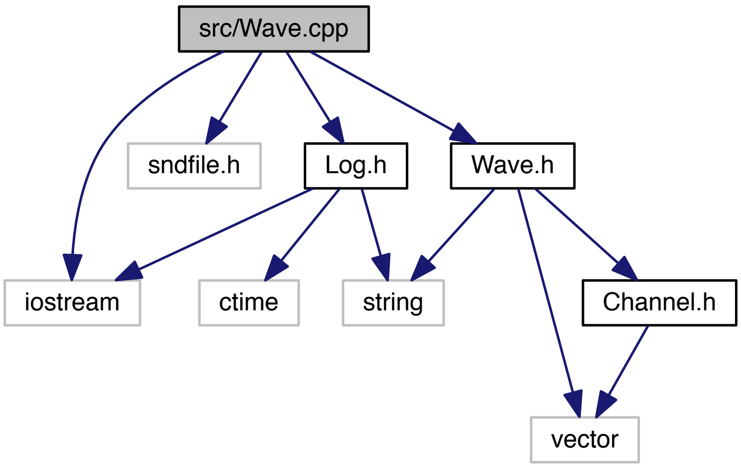
MIT License (see LICENSE file)

3.30 src/Wave.cpp File Reference

Wave file management via libsndfile.

```
#include <iostream>
#include <sndfile.h>
#include "Wave.h"
#include "Log.h"
```

Include dependency graph for Wave.cpp:



3.30.1 Detailed Description

Wave file management via libsndfile.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

5.2.2016

Copyright

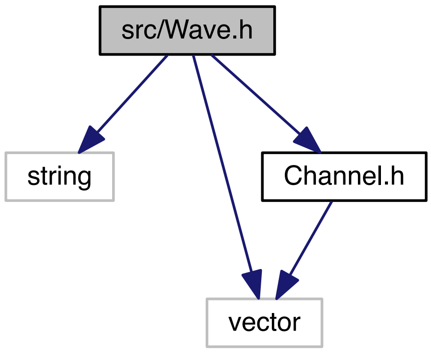
MIT License (see LICENSE file)

3.31 src/Wave.h File Reference

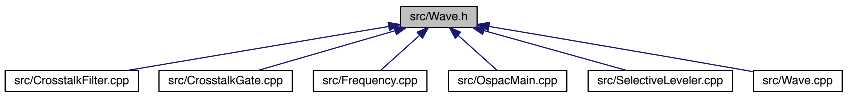
Wave file management via libsndfile.

```
#include <string>
#include <vector>
#include "Channel.h"
```

Include dependency graph for Wave.h:



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



Classes

- class Wave

  Wave-file loading and saving via libsndfile.

3.31.1 Detailed Description

Wave file management via libsndfile.

Author

Sebastian Ritterbusch `ospac@ritterbusch.de`

Version

1.0

Date

5.2.2016

Copyright

MIT License (see LICENSE file)

Index