RTMHA Library

2020a

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

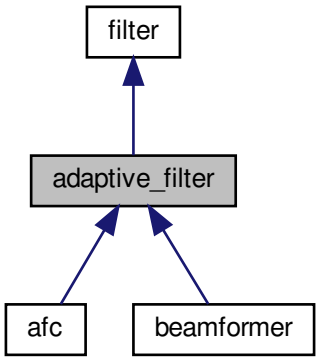
Class Documentation

4.1 adaptive_filter Class Reference

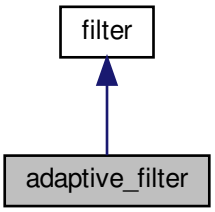
Adaptive Filter Class.

`#include <adaptive_filter.hpp>`

Inheritance diagram for adaptive_filter:



Collaboration diagram for adaptive_filter:



Public Member Functions

- adaptive_filter (float *adaptive_filter_taps, size_t adaptive_filter_tap_len, size_t max_frame_size, int adaptation↩ _type, float mu, float delta, float rho, float alpha, float beta, float p, float c, float power_estimate)

  Adaptive filter constructor.
- ~adaptive_filter ()

  Adaptive filter destructor.
- int update_taps (float *u_ref, float *e_ref, size_t ref_size)

  To update the taps of this adaptive filter based on the reference signals u_ref and e_ref.
- size_t get_max_frame_size ()

  Getting the maximum frame size.
- void get_params (float &mu, float &rho, float &delta, float &alpha, float &beta, float &p, float &c, int &adaptation↩ _type)

  Getting all parameters from this adaptive filter.
- void set_params (float mu, float rho, float delta, float alpha, float beta, float p, float c, int adaptation_type)

  Setting all parameters from this adaptive filter.

Protected Member Functions

- int get_adaptation_type ()

  A function to get the adaptation type.
- void get_step_size_weights_IPNLMS (float *taps, float *step_size_weights, float alpha, float beta, float delta, size_t tap_len)

  A function computing the step size control matrix for IPNLMS-l_0.
- void get_step_size_weights_SLMS (float *taps, float *step_size_weights, float p, float c, size_t tap_len)

  A function computing the step size control matrix for SLMS.

4.1.1 Detailed Description

Adaptive Filter Class.

This adaptive filter class implements several popular LMS-based algorithms including Modified LMS [Greenberg, 1998], IPNLMS-l_0 [Paleologu et al., 2010] and SLMS [Lee et al., 2017].

Definition at line 39 of file adaptive_filter.hpp.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 adaptive_filter()

```
adaptive_filter::adaptive_filter (
            float * adaptive_filter_taps,
            size_t adaptive_filter_tap_len,
            size_t max_frame_size,
            int adaptation_type,
            float mu,
            float delta,
            float rho,
            float alpha,
            float beta,
            float p,
            float c,
            float power_estimate )  [explicit]
```

Adaptive filter constructor.

Parameters

in	*adaptive_filter_taps*	The initial filter taps for adaptive filter
in	*adaptive_filter_tap_len*	The number of filter taps of adaptive filter
in	*max_frame_size*	The maximum processing frame size in adaptive filter
in	*adaptation_type*	-1: 0 y_hat, 0: stop adaptation, 1: Modified LMS, 2: IPNLMS-l_0, 3: SLMS
in	*mu*	The gradient descent step size (learning rate) for LMS-based algorithms
in	*delta*	A small positive number to prevent dividing zero
in	*rho*	The forgetting factor for power estimate
in	*alpha*	A number between -1 to 1 for different degrees of sparsity in IPNLMS-l_0
in	*beta*	A number between 0 to 500 for different degrees of sparsity in IPNLMS-l_0
in	*p*	A number between 0 to 2 for fitting different degrees of sparsity in SLMS
in	*c*	A small positive number for preventing stagnation in SLMS
in	*power_estimate*	An initial power estimate for adaptation

Definition at line 33 of file adaptive_filter.cpp.

4.1.3 Member Function Documentation

4.1.3.1 get_adaptation_type()

```
int adaptive_filter::get_adaptation_type ( )  [protected]
```

A function to get the adaptation type.

Returns

Adaptation type

Definition at line 164 of file adaptive_filter.cpp.

4.1.3.2 get_max_frame_size()

```
size_t adaptive_filter::get_max_frame_size ( )
```

Getting the maximum frame size.

Returns

Maximum frame size

Definition at line 128 of file adaptive_filter.cpp.

4.1.3.3 get_params()

```
void adaptive_filter::get_params (
            float & mu,
            float & rho,
            float & delta,
            float & alpha,
            float & beta,
            float & p,
            float & c,
            int & adaptation_type )
```

Getting all parameters from this adaptive filter.

Parameters

out	*mu*	The gradient descent step size (learning rate) for LMS-based algorithms
out	*rho*	The forgetting factor for power estimate
out	*delta*	A small positive number to prevent dividing zero
out	*alpha*	A number between -1 to 1 for different degrees of sparsity in IPNLMS-l_0
out	*beta*	A number between 0 to 500 for different degrees of sparsity in IPNLMS-l_0
out	*p*	A number between 0 to 2 for fitting different degrees of sparsity in SLMS
out	*c*	A small positive number for preventing stagnation in SLMS
out	*adaptation_type*	-1: 0 y_hat, 0: stop adaptation, 1: Modified LMS, 2: IPNLMS-l_0, 3: SLMS

Definition at line 133 of file adaptive_filter.cpp.

4.1.3.4 get_step_size_weights_IPNLMS()

```
void adaptive_filter::get_step_size_weights_IPNLMS (
             float * taps,
             float * step_size_weights,
             float alpha,
             float beta,
             float delta,
             size_t tap_len )  [protected]
```

A function computing the step size control matrix for IPNLMS-l_0.

Parameters

in	*taps*	The current filter taps of the adaptive filter
out	*step_size_weights*	The step size control matrix (it is an 1-D array due to the diagonal matrix)
in	*alpha*	A number between -1 to 1 for different degrees of sparsity in IPNLMS-l_0
in	*beta*	A number between 0 to 500 for different degrees of sparsity in IPNLMS-l_0
in	*delta*	A small positive number to prevent dividing zero
in	*tap_len*	The number of taps of the adaptive filter

Definition at line 170 of file adaptive_filter.cpp.

4.1.3.5 get_step_size_weights_SLMS()

```
void adaptive_filter::get_step_size_weights_SLMS (
             float * taps,
             float * step_size_weights,
             float p,
             float c,
             size_t tap_len )  [protected]
```

A function computing the step size control matrix for SLMS.

Parameters

in	*taps*	The current filter taps of the adaptive filter
out	*step_size_weights*	The step size control matrix (it is an 1-D array due to the diagonal matrix)
in	*p*	A number between 0 to 2 for fitting different degrees of sparsity in SLMS
in	*c*	A small positive number for preventing stagnation in SLMS
in	*tap_len*	The number of taps of the adaptive filter

Definition at line 184 of file adaptive_filter.cpp.

4.1.3.6 set_params()

```
void adaptive_filter::set_params (
            float mu,
            float rho,
            float delta,
            float alpha,
            float beta,
            float p,
            float c,
            int adaptation_type )
```

Setting all parameters from this adaptive filter.

Parameters

in	*mu*	The gradient descent step size (learning rate) for LMS-based algorithms
in	*rho*	The forgetting factor for power estimate
in	*delta*	A small positive number to prevent dividing zero
in	*alpha*	A number between -1 to 1 for different degrees of sparsity in IPNLMS-l_0
in	*beta*	A number between 0 to 500 for different degrees of sparsity in IPNLMS-l_0
in	*p*	A number between 0 to 2 for fitting different degrees of sparsity in SLMS
in	*c*	A small positive number for preventing stagnation in SLMS
in	*adaptation_type*	-1: 0 y_hat, 0: stop adaptation, 1: Modified LMS, 2: IPNLMS-l_0, 3: SLMS

Definition at line 147 of file adaptive_filter.cpp.

4.1.3.7 update_taps()

```
int adaptive_filter::update_taps (
            float * u_ref,
            float * e_ref,
            size_t ref_size )
```

To update the taps of this adaptive filter based on the reference signals u_ref and e_ref.

Parameters

in	*u_ref*	A reference input signal for adaptation
in	*e_ref*	A reference error signal for adaptation
in	*ref_size*	The size of each reference signal (u_ref and e_ref have the same size)

Returns

A flag indicating the success of adaptation

Definition at line 70 of file adaptive_filter.cpp.

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

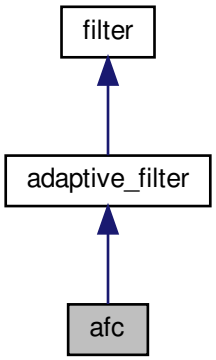
- adaptive_filter.hpp
- adaptive_filter.cpp

4.2 afc Class Reference

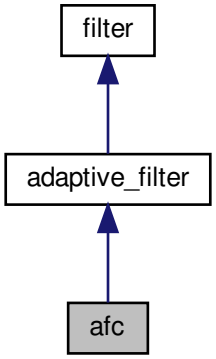
Adaptive Feedback Cancellation (AFC) Class.

```
#include <afc.hpp>
```

Inheritance diagram for afc:



Collaboration diagram for afc:



Public Member Functions

- afc (float *bandlimited_filter_taps, size_t bandlimited_filter_tap_len, float *prefilter_taps, size_t prefilter_tap_len, float *adaptive_filter_taps, size_t adaptive_filter_tap_len, size_t max_frame_size, int adaptation_type, float mu, float delta, float rho, float alpha, float beta, float p, float c, float power_estimate, size_t delay_len, int afc_on_off)

  AFC constructor.
- ~afc ()

  AFC destructor.
- int get_y_hat (float *y_hat, float *e, float *s, size_t ref_size)

  Getting y_hat signal (an estimated feedback signal)
- void get_delay (size_t &delay_len)

  Getting the length of delay line in samples.
- int set_delay (size_t delay_len)

  Setting the length of delay line in samples.
- void set_afc_on_off (int afc_on_off)

  Setting the ON/OFF for the AFC.
- void get_afc_on_off (int &afc_on_off)

  Getting the ON/OFF for the AFC.
- void reset (float *default_taps, size_t len)

  Reset the AFC filter to all zeros.

Additional Inherited Members

4.2.1 Detailed Description

Adaptive Feedback Cancellation (AFC) Class.

Under the FXLMS framework, this AFC class utilizes an adaptive filter to estimate the feedback signal, namely, y_hat.

Definition at line 40 of file afc.hpp.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 afc()

```
afc::afc (
            float * bandlimited_filter_taps,
            size_t bandlimited_filter_tap_len,
            float * prefilter_taps,
            size_t prefilter_tap_len,
            float * adaptive_filter_taps,
            size_t adaptive_filter_tap_len,
            size_t max_frame_size,
            int adaptation_type,
            float mu,
            float delta,
            float rho,
            float alpha,
            float beta,
            float p,
            float c,
            float power_estimate,
            size_t delay_len,
            int afc_on_off )  [explicit]
```

AFC constructor.

Parameters

in	*bandlimited_filter_taps*	The filter taps for bandlimited filter in AFC
in	*bandlimited_filter_tap_len*	The number of taps of bandlimited filter in AFC
in	*prefilter_taps*	The filter taps for whitening filter in AFC
in	*prefilter_tap_len*	The number of taps of prefilter in AFC
in	*adaptive_filter_taps*	The initial filter taps for adaptive filter in AFC
in	*adaptive_filter_tap_len*	The number of filter taps of adaptive filter in AFC
in	*max_frame_size*	The maximum processing frame size in adaptive filter
in	*adaptation_type*	The adaptation type for adaptive filter
in	*mu*	A parameter for adaptive filter
in	*delta*	A parameter for adaptive filter
in	*rho*	A parameter for adaptive filter
in	*alpha*	A parameter for adaptive filter
in	*beta*	A parameter for adaptive filter
in	*p*	A parameter for adaptive filter
in	*c*	A parameter for adaptive filter
in	*power_estimate*	A parameter for adaptive filter
in	*delay_len*	The number of delay in samples
in	*afc_on_off*	A flag to turn the AFC on (0: OFF, 1: ON)

See also

adaptive_filter

Definition at line 29 of file afc.cpp.

4.2.3 Member Function Documentation

4.2.3.1 get_afc_on_off()

```
void afc::get_afc_on_off (
            int & afc_on_off )
```

Getting the ON/OFF for the AFC.

Parameters

in	*afc_on_off*	A flag indicating the ON/OFF of AFC (False: OFF, True: ON)

Definition at line 120 of file afc.cpp.

4.2.3.2 get_delay()

```
void afc::get_delay (
            size_t & delay_len )
```

Getting the length of delay line in samples.

Parameters

out	*delay_len*	The number of delay in samples

Definition at line 91 of file afc.cpp.

4.2.3.3 get_y_hat()

```
int afc::get_y_hat (
            float * y_hat,
```

```
            float * e,
            float * s,
            size_t ref_size )
```

Getting y_hat signal (an estimated feedback signal)

Parameters

out	*y_hat*	An estimated feedback signal
in	*e*	An error signal for AFC (the output of hearing aid processing)
in	*s*	An input signal for AFC (the input of hearing aid processing)
in	*ref_size*	The size of each signal (e and s have the same size)

Returns

A flag indicating the success of getting correct y_hat according to the adaptation type

Definition at line 63 of file afc.cpp.

4.2.3.4 reset()

```
void afc::reset (
            float * default_taps,
            size_t len )
```

Reset the AFC filter to all zeros.

Parameters

in	*default_taps*	The default AFC filter
in	*len*	Length of the default AFC filter

Definition at line 126 of file afc.cpp.

4.2.3.5 set_afc_on_off()

```
void afc::set_afc_on_off (
            int afc_on_off )
```

Setting the ON/OFF for the AFC.

Parameters

in	*afc_on_off*	A flag to turn the AFC on (False: OFF, True: ON)

Definition at line 111 of file afc.cpp.

4.2.3.6 set_delay()

```
int afc::set_delay (
            size_t delay_len )
```

Setting the length of delay line in samples.

Parameters

in	*delay_len*	The number of delay in samples

Returns

A flag indicating the success of setting delay_len

Definition at line 97 of file afc.cpp.

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

- afc.hpp
- afc.cpp

4.3 array_file Class Reference

Array File Class.

`#include <array_file.hpp>`

Public Member Functions

- array_file (std::string path)
  Array file constructor.
- ~array_file ()
  Array file destructor.
- size_t get_len ()
  Getting the length of the array.
- float * get_ptr ()
  Getting the pointer which points to the array.

4.3.1 Detailed Description

Array File Class.

Reading a binary file into an array in single-precision floating-point format

Definition at line 33 of file array_file.hpp.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 array_file()

```
array_file::array_file (
            std::string path )
```

Array file constructor.

Parameters

path	The path of the binary file

Definition at line 32 of file array_file.cpp.

4.3.3 Member Function Documentation

4.3.3.1 get_len()

```
size_t array_file::get_len ( )
```

Getting the length of the array.

Returns

The length of the array

Definition at line 58 of file array_file.cpp.

4.3.3.2 get_ptr()

```
float * array_file::get_ptr ( )
```

Getting the pointer which points to the array.

Returns

The pointer which points to the array

Definition at line 63 of file array_file.cpp.

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

- array_file.hpp
- array_file.cpp

4.4 AudioFile< T > Class Template Reference

Public Types

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

Public Member Functions

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

Public Attributes

- AudioBuffer samples

4.4.1 Detailed Description

template<class T>
class AudioFile< T >

Definition at line 47 of file AudioFile.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 AudioFile()

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

Constructor

Definition at line 54 of file AudioFile.cpp.

4.4.3 Member Function Documentation

4.4.3.1 getBitDepth()

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

the bit depth of each sample

Definition at line 93 of file AudioFile.cpp.

4.4.3.2 getLengthInSeconds()

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

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

Definition at line 110 of file AudioFile.cpp.

4.4.3.3 getNumChannels()

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

the number of audio channels in the buffer

Definition at line 72 of file AudioFile.cpp.

4.4.3.4 getNumSamplesPerChannel()

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

the number of samples per channel

Definition at line 100 of file AudioFile.cpp.

4.4.3.5 getSampleRate()

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

the sample rate

Definition at line 65 of file AudioFile.cpp.

4.4.3.6 isMono()

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

true if the audio file is mono

Definition at line 79 of file AudioFile.cpp.

4.4.3.7 isStereo()

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

true if the audio file is stereo

Definition at line 86 of file AudioFile.cpp.

4.4.3.8 load()

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

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

Definition at line 221 of file AudioFile.cpp.

4.4.3.9 printSummary()

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

Prints a summary of the audio file to the console

Definition at line 117 of file AudioFile.cpp.

4.4.3.10 save()

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

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

Definition at line 525 of file AudioFile.cpp.

4.4.3.11 setAudioBuffer()

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

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

Definition at line 130 of file AudioFile.cpp.

4.4.3.12 setAudioBufferSize()

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

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

Definition at line 162 of file AudioFile.cpp.

4.4.3.13 setBitDepth()

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

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

Definition at line 207 of file AudioFile.cpp.

4.4.3.14 setNumChannels()

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

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

Definition at line 186 of file AudioFile.cpp.

4.4.3.15 setNumSamplesPerChannel()

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

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

Definition at line 170 of file AudioFile.cpp.

4.4.3.16 setSampleRate()

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

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

Definition at line 214 of file AudioFile.cpp.

4.4.4 Member Data Documentation

4.4.4.1 samples

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

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

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

Definition at line 126 of file AudioFile.h.

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

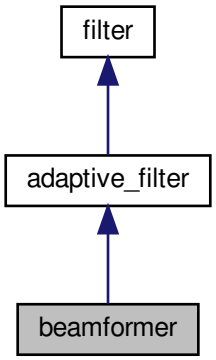
- AudioFile.h
- AudioFile.cpp

4.5 beamformer Class Reference

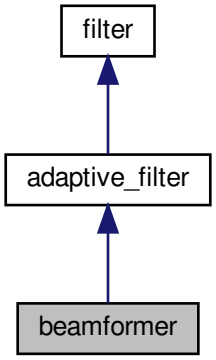
Beamformer Class.

`#include <beamformer.hpp>`

Inheritance diagram for beamformer:



Collaboration diagram for beamformer:



Public Member Functions

- beamformer (size_t delay_len, float *adaptive_filter_taps, size_t adaptive_filter_tap_len, size_t max_frame_size, int adaptation_type, float mu, float delta, float rho, float alpha, float beta, float p, float c, float power_estimate, int bf_on_off, int bf_nc_on_off, int bf_amc_on_off, float nc_thr, float amc_thr, float amc_forgetting_factor)

  Beamformer constructor.
- ~beamformer ()

  Beamformer destructor.
- void get_e (float *e_l, float *e_r, const float *x_l, const float *x_r, size_t ref_size)

  Getting e signal (the output signal of this beamformer)
- int update_bf_taps (size_t ref_size)

  Update the taps of the beamformer.
- void get_bf_params (int &bf_on_off, int &bf_nc_on_off, int &bf_amc_on_off, float &nc_thr, float &amc_thr, float &amc_forgetting_factor)

  Getting all parameters from this beamformer.
- void set_bf_params (int bf_on_off, int bf_nc_on_off, int bf_amc_on_off, float nc_thr, float amc_thr, float amc_forgetting_factor)

  Setting all parameters from this beamformer.

Additional Inherited Members

4.5.1 Detailed Description

Beamformer Class.

This beamformer class implements the generalized sidelobe canceller (GSC) using SLMS [Lee et al., IHCON 2018].

Definition at line 39 of file beamformer.hpp.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 beamformer()

```
beamformer::beamformer (
            size_t delay_len,
            float * adaptive_filter_taps,
            size_t adaptive_filter_tap_len,
            size_t max_frame_size,
            int adaptation_type,
            float mu,
            float delta,
            float rho,
            float alpha,
            float beta,
```

```
float p,
float c,
float power_estimate,
int bf_on_off,
int bf_nc_on_off,
int bf_amc_on_off,
float nc_thr,
float amc_thr,
float amc_forgetting_factor )  [explicit]
```

Beamformer constructor.

Parameters

in	*delay_len*	The length of delay line in samples for beamformer
in	*adaptive_filter_taps*	The initial filter taps for adaptive filter in beamformer
in	*adaptive_filter_tap_len*	The number of filter taps of adaptive filter in beamformer
in	*max_frame_size*	The maximum processing frame size in adaptive filter
in	*adaptation_type*	The adaptation type for adaptive filter
in	*mu*	A parameter for adaptive filter
in	*delta*	A parameter for adaptive filter
in	*rho*	A parameter for adaptive filter
in	*alpha*	A parameter for adaptive filter
in	*beta*	A parameter for adaptive filter
in	*p*	A parameter for adaptive filter
in	*c*	A parameter for adaptive filter
in	*power_estimate*	A parameter for adaptive filter
in	*bf_on_off*	A flag for enabling beamformer
in	*bf_nc_on_off*	A flag for enabling norm-constrained adaptation
in	*bf_amc_on_off*	A flag for enabling adaptation mode controller
in	*nc_thr*	A threshold for the norm-constrained adaptation
in	*amc_thr*	A threshold for the adaptation mode controller

See also

adaptive_filter

Definition at line 32 of file beamformer.cpp.

4.5.3 Member Function Documentation

4.5.3.1 get_bf_params()

```
void beamformer::get_bf_params (
            int & bf_on_off,
            int & bf_nc_on_off,
            int & bf_amc_on_off,
            float & nc_thr,
            float & amc_thr,
            float & amc_forgetting_factor )
```

Getting all parameters from this beamformer.

Parameters

out	*bf_on_off*	A flag to turn the beamformer on (0: OFF, 1: ON)
out	*bf_nc_on_off*	A flag to turn the norm-constrained adaptation on (0: OFF, 1: ON)
out	*bf_amc_on_off*	A flag to turn the adaptation mode controller on (0: OFF, 1: ON)
out	*nc_thr*	A threshold for the norm-constrained adaptation
out	*amc_thr*	A threshold for the adaptation mode controller
out	*amc_forgetting_factor*	A forgetting factor to compute the signal power

Definition at line 142 of file beamformer.cpp.

4.5.3.2 get_e()

```
void beamformer::get_e (
            float * e_l,
            float * e_r,
            const float * x_l,
            const float * x_r,
            size_t ref_size )
```

Getting e signal (the output signal of this beamformer)

Parameters

out	*e_l*	The output signal of this beamformer on the left
out	*e_r*	The output signal of this beamformer on the right
in	*x_l*	The input signal from the left channel
in	*x_r*	the input signal from the right channel
in	*ref_size*	The size of each input signal (x_l and x_r have the same size)

Definition at line 70 of file beamformer.cpp.

4.5.3.3 set_bf_params()

```
void beamformer::set_bf_params (
            int bf_on_off,
            int bf_nc_on_off,
            int bf_amc_on_off,
            float nc_thr,
            float amc_thr,
            float amc_forgetting_factor )
```

Setting all parameters from this beamformer.

Parameters

in	*bf_on_off*	A flag to turn the beamformer on (0: OFF, 1: ON)
in	*bf_nc_on_off*	A flag to turn the norm-constrained adaptation on (0: OFF, 1: ON)
in	*bf_amc_on_off*	A flag to turn the adaptation mode controller on (0: OFF, 1: ON)
in	*nc_thr*	A threshold for the norm-constrained adaptation
in	*amc_thr*	A threshold for the adaptation mode controller
in	*amc_forgetting_factor*	A forgetting factor to compute the signal power

Definition at line 153 of file beamformer.cpp.

4.5.3.4 update_bf_taps()

```
int beamformer::update_bf_taps (
            size_t ref_size )
```

Update the taps of the beamformer.

Parameters

in	*ref_size*	The size of each input signal (x_l and x_r have the same size)

Returns

A flag indicating the success of adaptation in adaptive filter

Definition at line 102 of file beamformer.cpp.

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

- beamformer.hpp
- beamformer.cpp

4.6 circular_buffer Class Reference

Circular Buffer Class.

`#include <circular_buffer.hpp>`

Public Member Functions

- circular_buffer (size_t size, float reset)

  Circular buffer constructor.
- ~circular_buffer ()

  Default destructor.
- void set (const float *item, size_t buf_size)

  This is the set command for the circular buffer.
- void get (float *data, size_t buf_size)

  This is the get function for the circular buffer.
- void delay_block (float *data, size_t buf_size, size_t delay)

  This is the get function for the circular buffer with a delay.
- void reset ()

  This is the reset command for circular buffer. It resets all of the values in the buffer to the default value the user entered in the constructor.
- size_t size () const

  Function to get the size of the buffer.

Public Attributes

- float * **buf_**
- std::atomic< size_t > **head_**
- size_t **size_**
- size_t **mask_**
- float **reset_**

4.6.1 Detailed Description

Circular Buffer Class.

Definition at line 36 of file circular_buffer.hpp.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 circular_buffer()

```
circular_buffer::circular_buffer (
            size_t size,
            float reset )  [explicit]
```

Circular buffer constructor.

Parameters

in	*size*	The maximum size you would want your circular buffer to be
in	*reset*	The value you want to reset all of the values in the circular buffer to

Definition at line 34 of file circular_buffer.cpp.

4.6.3 Member Function Documentation

4.6.3.1 delay_block()

```
void circular_buffer::delay_block (
            float * data,
            size_t buf_size,
            size_t delay )
```

This is the get function for the circular buffer with a delay.

Parameters

out	*data*	A buffer to put your data in.
in	*buf_size*	The amount of data you want from the circular buffer
in	*delay*	The delay you want form the delay block in terms of time step

Definition at line 87 of file circular_buffer.cpp.

4.6.3.2 get()

```
void circular_buffer::get (
            float * data,
            size_t buf_size )
```

This is the get function for the circular buffer.

Parameters

out	*data*	A buffer to put your data in.
in	*buf_size*	The amount of data you want from the circular buffer

Definition at line 71 of file circular_buffer.cpp.

4.6.3.3 set()

```
void circular_buffer::set (
            const float * item,
            size_t buf_size )
```

This is the set command for the circular buffer.

Parameters

in	*item*	The buffer of data you want to put in the circular buffer
in	*buf_size*	The size of the buffer.

Definition at line 52 of file circular_buffer.cpp.

4.6.3.4 size()

```
size_t circular_buffer::size ( ) const
```

Function to get the size of the buffer.

Returns

The size of the circular buffer which will be a power of 2

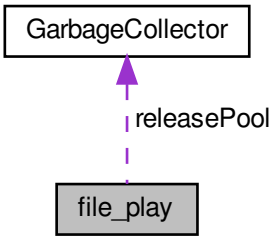
Definition at line 111 of file circular_buffer.cpp.

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

- circular_buffer.hpp
- circular_buffer.cpp

4.7 file_play Class Reference

Collaboration diagram for file_play:



Classes

- struct playfile_param_t

Public Member Functions

- void **rtmha_play** (int num_sample, float ∗out, int channel)
- void **set_params** (const char ∗, int, int, int)

Public Attributes

- std::string **rootPath**
- std::shared_ptr< playfile_param_t > **currentParam**
- GarbageCollector **releasePool**

4.7.1 Detailed Description

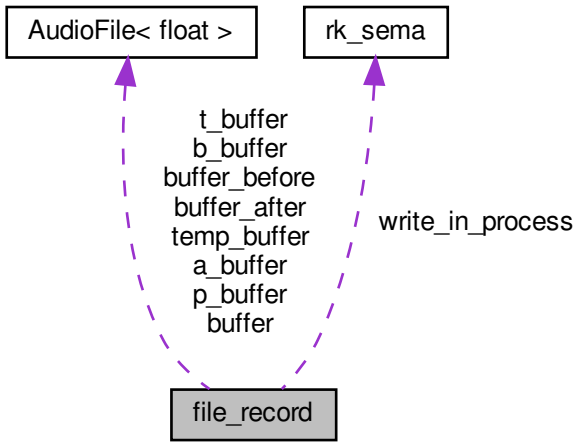
Definition at line 7 of file playfile.h.

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

- playfile.h
- playfile.cpp

4.8 file_record Class Reference

Collaboration diagram for file_record:



Public Member Functions

- **file_record** (int freq=48000, std::string file_="sample.wav", float seconds=5)
- void **rtmha_record** (int num_sample, float ∗in, int channel)
- void **record_before** (int num_sample, float ∗in, int channel)
- void **record_after** (int num_sample, float ∗in, int channel)
- void **set_params** (int start_, int stop_, float seconds, const char ∗file_)
- void **get_params** (float &seconds)
- void **write_to_file** ()

Public Attributes

- AudioFile< float >::AudioBuffer **buffer**
- AudioFile< float >::AudioBuffer **temp_buffer**
- AudioFile< float >::AudioBuffer **buffer_before**
- AudioFile< float >::AudioBuffer **buffer_after**
- std::string **rootPath**
- std::string **file**
- std::atomic< bool > **write**
- AudioFile< float >::AudioBuffer ∗ **p_buffer** =0
- AudioFile< float >::AudioBuffer ∗ **t_buffer** =0
- AudioFile< float >::AudioBuffer ∗ **b_buffer** =&buffer_before
- AudioFile< float >::AudioBuffer ∗ **a_buffer** =&buffer_after

- int **reset**
- int **start** =0
- int **stop** =0
- int **start_before**
- int **start_after** =0
- int **stop_before** =0
- int **stop_after**
- int **record**
- int **repeat**
- std::string **file_path**
- int **current_position_l**
- int **current_position_r**
- int **current_before_l**
- int **current_before_r**
- int **current_after_l**
- int **current_after_r**
- int **sampling_freq**
- int **numSamples**
- int **numSamples_before**
- int **numSamples_after**
- int **finish**
- int **mono**
- int **t**
- rk_sema * **write_in_process**
- std::thread * **record_thread**

4.8.1 Detailed Description

Definition at line 18 of file file_record.h.

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

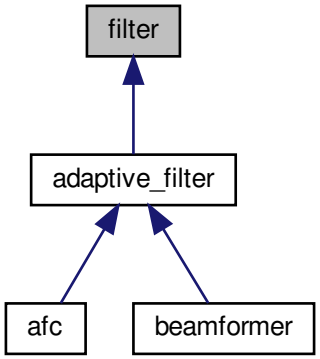
- file_record.h
- file_record.cpp

4.9 filter Class Reference

Filter Class.

`#include <filter.hpp>`

Inheritance diagram for filter:



Public Member Functions

- filter (float *taps, size_t tap_size, circular_buffer *cir_buf, size_t max_buf_size)

  Filter constructor.
- ~filter ()

  Filter destructor.
- int set_taps (const float *taps, size_t buf_size)

  Setting the filter taps.
- int get_taps (float *taps, size_t buf_size)

  Getting the filter taps.
- void cirfir (float *data_in, float *data_out, size_t num_samp)

  Getting the output of this FIR filter by performing frame-based convolution.
- size_t get_size ()

  Getting the number of taps of this FIR filter.
- void cirfir (float *data_out, size_t num_samp)

  Frame-based convolution for FIR filtering.

4.9.1 Detailed Description

Filter Class.

This filter class implements the FIR filter

Definition at line 39 of file filter.hpp.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 filter()

```
filter::filter (
            float * taps,
            size_t tap_size,
            circular_buffer * cir_buf,
            size_t max_buf_size )  [explicit]
```

Filter constructor.

Parameters

in	*taps*	The filter taps for this FIR filter
in	*tap_size*	The number of taps of this FIR filter
in	*cir_buf*	The circular buffer for this FIR filter to perform frame-based convolution
in	*max_buf_size*	The maximum size of circular buffer you need to specify if there is no circular buffer given in cir_buf

Definition at line 31 of file filter.cpp.

4.9.3 Member Function Documentation

4.9.3.1 cirfir() [1/2]

```
void filter::cirfir (
            float * data_in,
            float * data_out,
            size_t num_samp )
```

Getting the output of this FIR filter by performing frame-based convolution.

Parameters

in	*data_in*	The input signal
out	*data_out*	The output signal
	num_samp	The size of input and output signal (data_in and data_out should have the same size)

Definition at line 96 of file filter.cpp.

4.9.3.2 cirfir() [2/2]

```
void filter::cirfir (
            float * data_out,
            size_t num_samp )
```

Frame-based convolution for FIR filtering.

Parameters

out	*data_out*	The output signal
in	*num_samp*	The size of input and output signal

Definition at line 107 of file filter.cpp.

4.9.3.3 get_size()

```
size_t filter::get_size ( )
```

Getting the number of taps of this FIR filter.

Returns

The number of taps of this FIR filter

Definition at line 102 of file filter.cpp.

4.9.3.4 get_taps()

```
int filter::get_taps (
            float * taps,
            size_t buf_size )
```

Getting the filter taps.

Parameters

out	*taps*	The filter taps (1-D array)
in	*buf_size*	The size of the filter taps (this should be the same as tap_size passed in constructor)

Returns

A flag indicating the success of getting the filter taps

Definition at line 84 of file filter.cpp.

4.9.3.5 set_taps()

```
int filter::set_taps (
            const float * taps,
            size_t buf_size )
```

Setting the filter taps.

Parameters

in	*taps*	The filter taps (an 1-D array)
in	*buf_size*	The size of the filter taps (this should be the same as tap_size passed in constructor)

Returns

A flag indicating the success of setting the filter taps

Definition at line 68 of file filter.cpp.

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

- filter.hpp
- filter.cpp

4.10 fir_formii Class Reference

Filter Class.

```
#include <fir_formii.h>
```

Public Member Functions

- fir_formii (float *taps, size_t tap_size)

  Filter constructor.
- ~fir_formii ()

  Filter destructor.
- void set_taps (const float *taps, size_t tap_size)

  Setting the filter taps.
- void get_taps (float *taps, size_t &tap_size)

  Getting the filter taps.
- void process (const float *data_in, float *data_out, size_t num_samp)

  Getting the output of this FIR filter by performing frame-based convolution.
- size_t get_size ()

  Getting the number of taps of this FIR filter.

4.10.1 Detailed Description

Filter Class.

This filter class implements the FIR filter

Definition at line 41 of file fir_formii.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 fir_formii()

```
fir_formii::fir_formii (
            float * taps,
            size_t tap_size )  [explicit]
```

Filter constructor.

Parameters

in	*taps*	The filter taps for this FIR filter
in	*tap_size*	The number of taps of this FIR filter
in	*cir_buf*	The circular buffer for this FIR filter to perform frame-based convolution
in	*max_buf_size*	The maximum size of circular buffer you need to specify if there is no circular buffer given in cir_buf

Definition at line 29 of file fir_formii.cpp.

4.10.3 Member Function Documentation

4.10.3.1 get_size()

```
size_t fir_formii::get_size ( )
```

Getting the number of taps of this FIR filter.

Returns

The number of taps of this FIR filter

Definition at line 97 of file fir_formii.cpp.

4.10.3.2 get_taps()

```
void fir_formii::get_taps (
            float * taps,
            size_t & tap_size )
```

Getting the filter taps.

Parameters

out	*taps*	The filter taps (1-D array)
in	*buf_size*	The size of the filter taps (this should be the same as tap_size passed in constructor)

Returns

A flag indicating the success of getting the filter taps

Definition at line 65 of file fir_formii.cpp.

4.10.3.3 process()

```
void fir_formii::process (
            const float * data_in,
            float * data_out,
            size_t num_samp )
```

Getting the output of this FIR filter by performing frame-based convolution.

Parameters

in	*data_in*	The input signal
out	*data_out*	The output signal
	num_samp	The size of input and output signal (data_in and data_out should have the same size)

Definition at line 73 of file fir_formii.cpp.

4.10.3.4 set_taps()

```
void fir_formii::set_taps (
            const float * taps,
            size_t tap_size )
```

Setting the filter taps.

Parameters

in	*taps*	The filter taps (an 1-D array)
in	*buf_size*	The size of the filter taps (this should be the same as tap_size passed in constructor)

Returns

A flag indicating the success of setting the filter taps

Definition at line 49 of file fir_formii.cpp.

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

- fir_formii.h
- fir_formii.cpp

4.11 freping Class Reference

Freping Class.

```
#include <freping.hpp>
```

Public Member Functions

- freping (int allpass_chain_len, int frame_size, float alpha, float ∗window, int freping_on_off)

  Freping constructor.
- ∼freping ()

  Freping destructor.
- void get_params (float &alpha, int &freping_on_off)

  Get the parameter alpha.
- void set_params (float alpha, int freping_on_off)

  Set the parameter alpha.
- void freping_proc (float ∗in, float ∗out)

Protected Member Functions

- void overlap_add (const float ∗in, float ∗out)
- void allpass_chain (float ∗in, float ∗out, float alpha_, float coeff_)

  The all-pass chain which has the length of allpass_chain_len.
- void windowing (const float ∗in, float ∗out)

4.11.1 Detailed Description

Freping Class.

This freping class provides frequency warping feature in real-time. Please refer to the following paper for details. Ching-Hua Lee, Kuan-Lin Chen, fred harris, Bhaskar D. Rao, and Harinath Garudadri, "On mitigating acoustic feedback in hearing aids with frequency warping by all-pass networks," in Annual Conference of the International Speech Communication Association (Interspeech), 2019.

Definition at line 44 of file freping.hpp.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 freping()

```
freping::freping (
            int allpass_chain_len,
            int frame_size,
            float alpha,
            float * window,
            int freping_on_off )  [explicit]
```

Freping constructor.

Parameters

in	*allpass_chain_len*	The length of the all-pass chain
in	*frame_size*	The frame size of the caller (the real-time system)
in	*alpha*	The parameter to control the degree of frequency warping (1.0>=alpha>=-1.0)

As long as this is true we don't need circular buf function

Definition at line 29 of file freping.cpp.

4.11.3 Member Function Documentation

4.11.3.1 allpass_chain()

```
void freping::allpass_chain (
            float * in,
            float * out,
            float alpha_,
            float coeff_ )  [protected]
```

The all-pass chain which has the length of allpass_chain_len.

Parameters

in	*in*	The input frame which has the same size of the output frame (e.g., 128 samples)
out	*out*	The output frame (e.g., 128 samples)
in	*alpha_*	The parameter to control the degree of frequency warping (1.0>=alpha>=-1.0)
in	*coeff_*	a coefficient for the second IIR filter in the all-pass chain (a function of alpha_)

the fist stage of the all-pass chain

the second stage of the all-pass chain

Pointer swaping cost less than copying memory

the remaining stages of the all-pass chain

Pointer swaping cost less than copying memory

Definition at line 78 of file freping.cpp.

4.11.3.2 freping_proc()

```
void freping::freping_proc (
            float * in,
            float * out )
```

Get the output signal of freping, the output signal is warped according to the alpha parameter

Parameters

in	*in*	The input frame
out	*out*	The output frame

in_buf_->set(in,frame_size_); // Do we really need if frame_size is already a multiple of all_pass_len m_ = ++run_k_; /// Why is this useful?

in_buf_->get(x_buf_,buf_len_); this->overlap_framing(x_buf_,overlapped_frame_);

Pointer swaping cost less than copying memory

Definition at line 157 of file freping.cpp.

4.11.3.3 get_params()

```
void freping::get_params (
            float & alpha,
            int & freping_on_off )
```

Get the parameter alpha.

Parameters

out	*alpha*	The parameter to control the degree of frequency warping (1.0>=alpha>=-1.0)

Definition at line 133 of file freping.cpp.

4.11.3.4 overlap_add()

```
void freping::overlap_add (
            const float * in,
            float * out )  [protected]
```

Overlap-add method

Parameters

in	*in*	The overlapped frame which has the twice size of the output frame (e.g., 128 samples)
out	*out*	The recovered frame (e.g., 64 samples)

This should do the same thing

Definition at line 119 of file freping.cpp.

4.11.3.5 set_params()

```
void freping::set_params (
            float alpha,
            int freping_on_off )
```

Set the parameter alpha.

Parameters

in	*alpha*	The parameter to control the degree of frequency warping (1.0>=alpha>=-1.0)

Definition at line 139 of file freping.cpp.

4.11.3.6 windowing()

```
void freping::windowing (
            const float * in,
            float * out )  [protected]
```

Applying a window function such as Hamming window to the input frame

Parameters

in	*in*	The input frame which has the same size of the output frame (e.g., 128 samples)
out	*out*	The output frame (e.g., 128 samples)

Definition at line 72 of file freping.cpp.

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

- freping.hpp
- freping.cpp

4.12 GarbageCollector Class Reference

Public Member Functions

- template<typename T >
  void **add** (const std::shared_ptr< T > &object)
- void **release** ()

Public Attributes

- std::vector< std::shared_ptr< void > > **pool**
- std::mutex **m**

4.12.1 Detailed Description

Definition at line 36 of file GarbageCollector.hpp.

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

- GarbageCollector.hpp

4.13 libModule Class Reference

A template for library modules.

Public Member Functions

- libModule (...)

  libModule constructor
- ~libModule ()

  libModule destructor
- void set_param (...)

  Setting libModule parameters.
- void get_param (...)

  Getting libModule parameters.
- void process (...)

  Real-time processing inside libModule.

4.13.1 Detailed Description

A template for library modules.

Definition at line 16 of file example_template.cpp.

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

- example_template.cpp

4.14 noise_management Class Reference

Noise Management Class.

`#include <noise_management.hpp>`

Public Member Functions

- noise_management (int ntype, int stype, float sparam, float fsamp)
  Noise management constructor.
- ~noise_management ()
  Noise management destructor.
- void set_param (int ntype, int stype, float sparam)
  Setting all parameters in noise management.
- void get_param (int &ntype, int &stype, float &sparam)
  Getting all parameters in noise management.
- void speech_enhancement (float *data_in, size_t in_len, float *data_out)
  A function to perform speech enhancement.

4.14.1 Detailed Description

Noise Management Class.

Speech enhancement using peak and valley detection, noise estimation and spectral subtraction

Definition at line 16 of file noise_management.hpp.

4.14.2 Constructor & Destructor Documentation

4.14.2.1 noise_management()

```
noise_management::noise_management (
            int ntype,
            int stype,
            float sparam,
            float fsamp )  [explicit]
```

Noise management constructor.

Parameters

in	*ntype*	The type of noise estimation, 1: using limits on change (ref: Arslan et al.), 2: using the weighted averaging of Hirsch and Ehrlicher, 3: using MCRA of Cohen and Berdugo
in	*stype*	The type of spectral subtraction, 0: normal, 1: oversubtraction
in	*sparam*	A parameter for spectral subtraction
in	*fsamp*	The sampling rate

Definition at line 5 of file noise_management.cpp.

4.14.3 Member Function Documentation

4.14.3.1 get_param()

```
void noise_management::get_param (
            int & ntype,
            int & stype,
            float & sparam )
```

Getting all parameters in noise management.

Parameters

in	*ntype*	See constructor
in	*stype*	See constructor
in	*sparam*	See constructor

Definition at line 86 of file noise_management.cpp.

4.14.3.2 set_param()

```
void noise_management::set_param (
            int ntype,
            int stype,
            float sparam )
```

Setting all parameters in noise management.

Parameters

in	*ntype*	See constructor
in	*stype*	See constructor
in	*sparam*	See constructor

Definition at line 77 of file noise_management.cpp.

4.14.3.3 speech_enhancement()

```
void noise_management::speech_enhancement (
            float * data_in,
            size_t in_len,
            float * data_out )
```

A function to perform speech enhancement.

Parameters

in	*data_in*	The input signal
in	*in_len*	Length of the input signal
out	*data_out*	The output signal, i.e., the enhanced speech signal

Definition at line 95 of file noise_management.cpp.

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

- noise_management.hpp
- noise_management.cpp

4.15 peak_detect Class Reference

Peak Detector Class.

```
#include <peak_detect.hpp>
```

Public Member Functions

- peak_detect (float fsamp, float attack_time, float release_time)

  Peak detector constructor.
- ~peak_detect ()

  Peak detector destructor.
- void set_param (float attack_time, float release_time)

  Setting the parameters for peak detector (to have alpha and beta)
- void get_param (float &attack_time, float &release_time)

  Getting the parameters from peak detector (in terms of attach time and release time)
- void get_spl (float *data_in, size_t in_len, float *pdb_out)

  Getting the output from the peak detector in SPL.

4.15.1 Detailed Description

Peak Detector Class.

This peak detector implements the algorithm according to Eq. (8.1) in [James M. Kates, Digital hearing aids, Plural publishing, 2008].

Definition at line 21 of file peak_detect.hpp.

4.15.2 Constructor & Destructor Documentation

4.15.2.1 peak_detect()

```
peak_detect::peak_detect (
            float fsamp,
            float attack_time,
            float release_time )  [explicit]
```

Peak detector constructor.

Parameters

in	*fsamp*	The sampling rate of the system
in	*attack_time*	Attack time in milliseconds
in	*release_time*	Release time in milliseconds

Definition at line 4 of file peak_detect.cpp.

4.15.3 Member Function Documentation

4.15.3.1 get_param()

```
void peak_detect::get_param (
            float & attack_time,
            float & release_time )
```

Getting the parameters from peak detector (in terms of attach time and release time)

Parameters

out	*attack_time*	attack_time Attack time in milliseconds
out	*release_time*	release_time Release time in milliseconds

Definition at line 37 of file peak_detect.cpp.

4.15.3.2 get_spl()

```
void peak_detect::get_spl (
            float * data_in,
            size_t in_len,
            float * pdb_out )
```

Getting the output from the peak detector in SPL.

Parameters

in	*data_in*	The input signal
in	*in_len*	The size of the input signal
out	*pdb_out*	The output of peak detector in SPL

Definition at line 45 of file peak_detect.cpp.

4.15.3.3 set_param()

```
void peak_detect::set_param (
            float attack_time,
            float release_time )
```

Setting the parameters for peak detector (to have alpha and beta)

Parameters

in	*attack_time*	Attack time in milliseconds
in	*release_time*	Release time in milliseconds

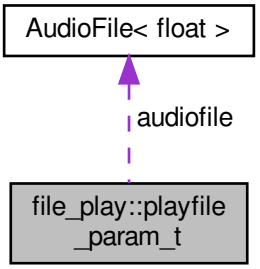
Definition at line 19 of file peak_detect.cpp.

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

- peak_detect.hpp
- peak_detect.cpp

4.16 file_play::playfile_param_t Struct Reference

Collaboration diagram for file_play::playfile_param_t:



Public Attributes

- int **reset**
- int **play**
- int **repeat**
- int **current_position_l**
- int **current_position_r**
- int **numSamples**
- bool **finish**
- int **mono**
- std::string **filename**
- AudioFile< float > * **audiofile**

4.16.1 Detailed Description

Definition at line 14 of file playfile.h.

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

- playfile.h

4.17 polyphase_hb_downsampler Class Reference

Public Member Functions

- **polyphase_hb_downsampler** (float *filter_taps, size_t num_taps, size_t max_frame_size)
- void **process** (float *in, float *out, size_t frame_size)

4.17.1 Detailed Description

Definition at line 13 of file polyphase_hb_downsampler.h.

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

- polyphase_hb_downsampler.h
- polyphase_hb_downsampler.cpp

4.18 polyphase_hb_upsampler Class Reference

Public Member Functions

- **polyphase_hb_upsampler** (float *filter_taps, size_t num_taps, size_t max_frame_size)
- void **process** (float *in, float *out, size_t frame_size)

4.18.1 Detailed Description

Definition at line 12 of file polyphase_hb_upsampler.h.

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

- polyphase_hb_upsampler.h
- polyphase_hb_upsampler.cpp

4.19 resample Class Reference

Resample Class.

`#include <resample.hpp>`

Public Member Functions

- resample (float *taps, size_t tap_size, size_t max_in_buf_size, int interp_factor, int deci_factor)

  Resample constructor.
- ~resample ()

  Resample destructor.
- void resamp (float *data_in, size_t in_size, float *data_out, size_t *out_size)

  Getting the resampled signal.

4.19.1 Detailed Description

Resample Class.

Resampling class implements L/M-fold resampling

Definition at line 15 of file resample.hpp.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 resample()

```
resample::resample (
            float * taps,
            size_t tap_size,
            size_t max_in_buf_size,
            int interp_factor,
            int deci_factor )  [explicit]
```

Resample constructor.

Parameters

in	*taps*	The filter taps of the lowpass filter (to reject images and prevent aliasing)
in	*tap_size*	The number of taps of the lowpass filter
in	*max_in_buf_size*	The maximum input buffer size
in	*interp_factor*	The interpolation factor L (to implement L-fold expander)
in	*deci_factor*	The decimation factor M (to implement M-fold decimator)

Definition at line 7 of file resample.cpp.

4.19.3 Member Function Documentation

4.19.3.1 resamp()

```
void resample::resamp (
            float * data_in,
            size_t in_size,
            float * data_out,
            size_t * out_size )
```

Getting the resampled signal.

Parameters

in	*data_in*	The signal in original sampling rate
in	*in_size*	The size of the original signal
out	*data_out*	The resampled signal
out	*out_size*	The size of the resampled signal

Definition at line 23 of file resample.cpp.

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

- resample.hpp
- resample.cpp

4.20 rk_sema Struct Reference

Public Attributes

- sem_t **sem**

4.20.1 Detailed Description

Definition at line 8 of file sema.hpp.

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

- sema.hpp

4.21 tenband_filterbank Class Reference

The Ten Band Filter bank is a 10 Band Multirate Filter Bank with its center frequencies located at 250Hz, 500Hz, 750Hz, 1kHz, 1.5kHz, 2kHz, 3kHz, 4kHz, 6kHz, 8kHz.

`#include <tenband_filterbank.h>`

Public Member Functions

- tenband_filterbank (size_t max_frame_size, bool aligned)

  The consturctor for the Ten Band Filter bank.
- ~tenband_filterbank ()

  The desturctor for the Ten Band Filter bank.
- void process (float *in, float **out, size_t frame_size)

  This function decomposes the incoming frame of data into the 10 subbands.
- void set (bool aligned)

  The compensation for the various group delay can be toggled on and off live.
- void get (bool &aligned)

  Used to read the current status of the aligned variable.

4.21.1 Detailed Description

The Ten Band Filter bank is a 10 Band Multirate Filter Bank with its center frequencies located at 250Hz, 500Hz, 750Hz, 1kHz, 1.5kHz, 2kHz, 3kHz, 4kHz, 6kHz, 8kHz.

Definition at line 44 of file tenband_filterbank.h.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 tenband_filterbank()

```
tenband_filterbank::tenband_filterbank (
            size_t max_frame_size,
            bool aligned )
```

The consturctor for the Ten Band Filter bank.

Parameters

max_frame_size	[in] - The maximum size of buffer that would need to be processed at any time step.
aligned	[in] - Enabling the delay blocks to compensate for the various group delays.

Definition at line 31 of file tenband_filterbank.cpp.

4.21.3 Member Function Documentation

4.21.3.1 get()

```
void tenband_filterbank::get (
            bool & aligned )
```

Used to read the current status of the aligned variable.

Parameters

out	*aligned*	Grabbing the current status of the aligned variable.

Definition at line 200 of file tenband_filterbank.cpp.

4.21.3.2 process()

```
void tenband_filterbank::process (
            float * in,
            float ** out,
            size_t frame_size )
```

This function decomposes the incoming frame of data into the 10 subbands.

Parameters

in	*in*	An array of data the size of frame_size.
out	*out*	A 2D 10 by frame_size array of data for the 10 subbands
in	*frame_size*	The amount of data contained in the incoming array.

Definition at line 150 of file tenband_filterbank.cpp.

4.21.3.3 set()

```
void tenband_filterbank::set (
            bool aligned )
```

The compensation for the various group delay can be toggled on and off live.

Parameters

in	*aligned*	Enabling the delay blocks to compensate for the various group delays.

Definition at line 197 of file tenband_filterbank.cpp.

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

- tenband_filterbank.h
- tenband_filterbank.cpp

4.22 wdrc Class Reference

Wide Dynamic Range Compression (WDRC) Class.

```
#include <wdrc.hpp>
```

Public Member Functions

- wdrc (float gain50, float gain80, float knee_low, float mpo_limit)
  wdrc constructor
- ~wdrc ()
  wdrc destructor
- void set_param (float gain50, float gain80, float knee_low, float mpo_limit)
  Setting WDRC parameters.
- void get_param (float &gain50, float &gain80, float &knee_low, float &mpo_limit)
  Getting WDRC parameters.
- void process (float *input, float *pdb, size_t in_len, float *output)
  Perform WDRC.

4.22.1 Detailed Description

Wide Dynamic Range Compression (WDRC) Class.

Applying WDRC to a subband signal from an analysis filterbank

Definition at line 18 of file wdrc.hpp.

4.22.2 Constructor & Destructor Documentation

4.22.2.1 wdrc()

```
wdrc::wdrc (
            float gain50,
            float gain80,
            float knee_low,
            float mpo_limit )  [explicit]
```

wdrc constructor

Parameters

in	*gain50*	Gain at 50 dB SPL of input level
in	*gain80*	Gain at 80 dB SPL of input level
in	*knee_low*	Lower knee-point
in	*mpo_limit*	Maximum power output (MPO)

Definition at line 3 of file wdrc.cpp.

4.22.3 Member Function Documentation

4.22.3.1 get_param()

```
void wdrc::get_param (
            float & gain50,
            float & gain80,
            float & knee_low,
            float & mpo_limit )
```

Getting WDRC parameters.

Parameters

out	*gain50*	Gain at 50 dB SPL of input level
out	*gain80*	Gain at 80 dB SPL of input level
out	*knee_low*	Lower knee-point
out	*mpo_limit*	MPO

Definition at line 43 of file wdrc.cpp.

4.22.3.2 process()

```
void wdrc::process (
            float * input,
            float * pdb,
            size_t in_len,
            float * output )
```

Perform WDRC.

The peak detector output in dB SPL is needed as one of the inputs. The gain at 50 and 80 dB SPL is specified for the frequency sub-band, along with the lower and upper kneepoints in dB SPL. The compressor is linear below the lower kneepoint and applies compression limiting above the upper kneepoint

Parameters

in	*input*	The input signal (1-D array)
in	*pdb*	The output from the peak detector in SPL, i.e., the output from get_spl member function in peak_detect class
in	*in_len*	Length of the input signal
out	*output*	Pointer to a signal (1-D array) where the compressed output of the subband signal will be written, i.e., the output of WDRC

See also

peak_detect

Definition at line 52 of file wdrc.cpp.

4.22.3.3 set_param()

```
void wdrc::set_param (
            float gain50,
            float gain80,
            float knee_low,
            float mpo_limit )
```

Setting WDRC parameters.

Parameters

in	*gain50*	Gain at 50 dB SPL of input level
in	*gain80*	Gain at 80 dB SPL of input level
in	*knee_low*	Lower knee-point
in	*mpo_limit*	MPO

Definition at line 25 of file wdrc.cpp.

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

- wdrc.hpp
- wdrc.cpp

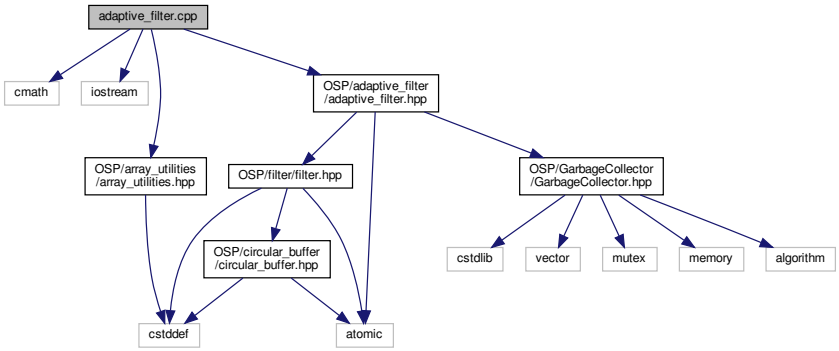
Chapter 5

File Documentation

5.1 adaptive_filter.cpp File Reference

```
#include <cmath>
#include <iostream>
#include <OSP/array_utilities/array_utilities.hpp>
#include <OSP/adaptive_filter/adaptive_filter.hpp>
```

Include dependency graph for adaptive_filter.cpp:



5.1.1 Detailed Description

Author

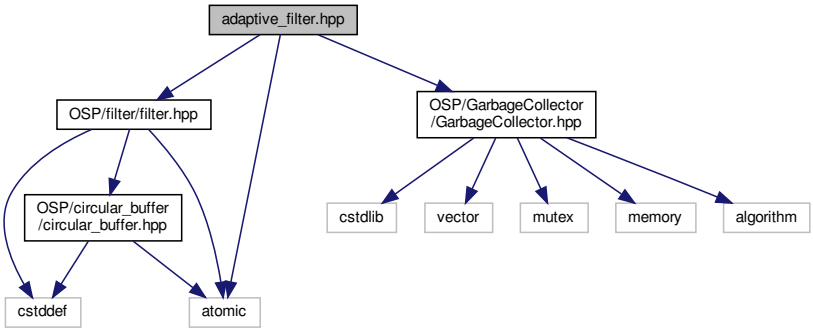
Open Speech Platform (OSP) Team, UCSD

Copyright

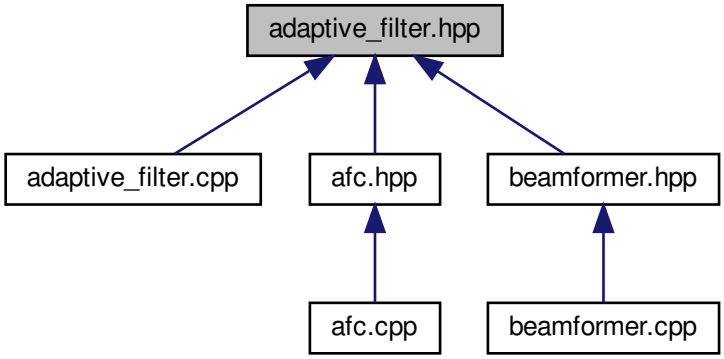
5.2 adaptive_filter.hpp File Reference

```
#include <OSP/filter/filter.hpp>
#include <atomic>
#include <OSP/GarbageCollector/GarbageCollector.hpp>
```

Include dependency graph for adaptive_filter.hpp:



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



Classes

- class adaptive_filter

  Adaptive Filter Class.

5.2.1 Detailed Description

Author

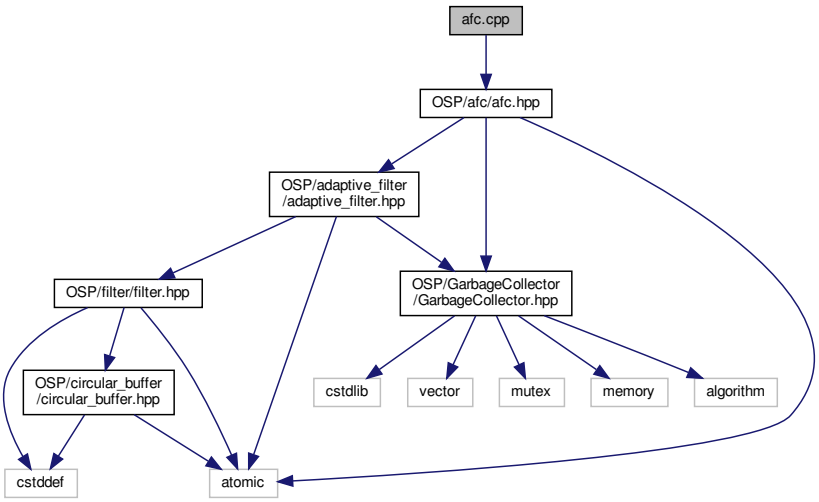
Open Speech Platform (OSP) Team, UCSD

Copyright

5.3 afc.cpp File Reference

```
#include <OSP/afc/afc.hpp>
```

Include dependency graph for afc.cpp:



5.3.1 Detailed Description

Author

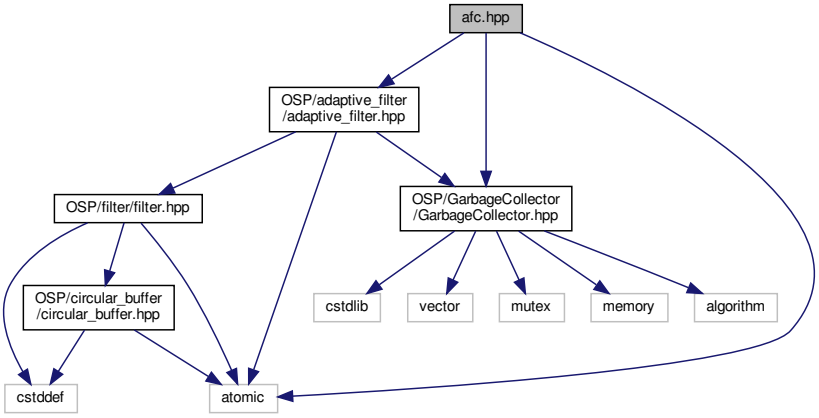
Open Speech Platform (OSP) Team, UCSD

Copyright

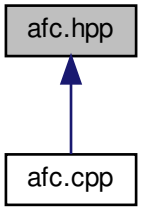
5.4 afc.hpp File Reference

```
#include <OSP/adaptive_filter/adaptive_filter.hpp>
#include <atomic>
#include <OSP/GarbageCollector/GarbageCollector.hpp>
```

Include dependency graph for afc.hpp:



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



Classes

- class afc

  Adaptive Feedback Cancellation (AFC) Class.

Macros

- #define **MAX_DELAY_LEN** 256

5.4.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.5 afc_init_filter.h File Reference

Macros

- #define **AFC_INIT_FILTER_SIZE** 160

Variables

- float **afc_init_filter** [AFC_INIT_FILTER_SIZE]

5.5.1 Detailed Description

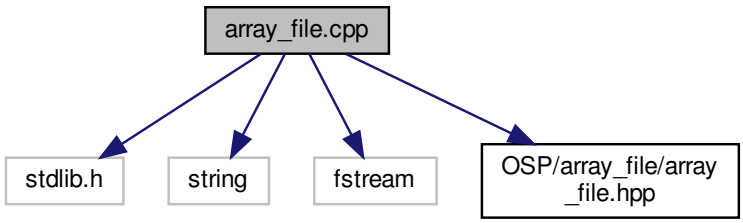
Author

Open Speech Platform (OSP) Team, UCSD

5.6 array_file.cpp File Reference

```
#include <stdlib.h>
#include <string>
#include <fstream>
#include <OSP/array_file/array_file.hpp>
```

Include dependency graph for array_file.cpp:



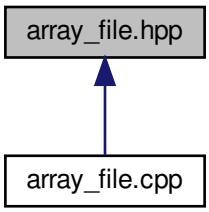
5.6.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

5.7 array_file.hpp File Reference

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



Classes

- class array_file

  Array File Class.

5.7.1 Detailed Description

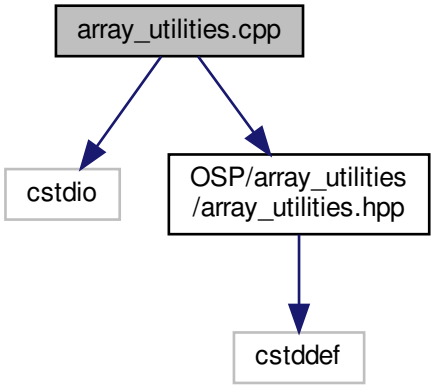
Author

Open Speech Platform (OSP) Team, UCSD

5.8 array_utilities.cpp File Reference

```
#include <cstdio>
#include <OSP/array_utilities/array_utilities.hpp>
```

Include dependency graph for array_utilities.cpp:



Functions

- void array_flip (float *arr, size_t len)

  Function to reverse an array.

- float array_sum (const float *arr, size_t len)

  Function to calculate the sum of an array.
- float array_dot_product (const float *in1, const float *in2, size_t len)

  Function to calculate the dot-product of two 1-D vectors/arrays.
- void array_right_shift (float *arr, size_t len)

  Function to right shift an array by one place. Left most value will be replaced by zero.
- void array_multiply_const (float *arr, float constant, size_t len)

  Function to multiply each element of an array by a scalar constant.
- void array_add_const (float *arr, float constant, size_t len)

  Function to add a scalar constant to each element of an array.
- void array_add_array (float *in1, const float *in2, size_t len)

  Function to do element wise addition of two arrays.
- void array_subtract_array (float *in1, const float *in2, size_t len)

  Function to do element wise subtraction of two arrays.
- void array_element_multiply_array (float *in1, const float *in2, size_t len)

  Function to do element wise multiplication of two arrays.
- void array_element_divide_array (float *in1, const float *in2, size_t len)

  Function to do element wise division of two arrays.
- float array_min (const float *arr, size_t len)

  Function to return the minimum of the elements of an array.
- float array_mean (float *arr, size_t len)

  Function to calculate the mean of the elements of an array.
- void array_square (const float *in, float *out, size_t len)

  Function to populate the output array with square of the elements of an input array.
- float array_mean_square (const float *arr, size_t len)

  Function to calculate the mean square of the elements of an array.
- void array_print (const char *str, float *arr, size_t len)

  Function to print an array for debugging.

5.8.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.8.2 Function Documentation

5.8.2.1 array_add_array()

```
void array_add_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise addition of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 105 of file array_utilities.cpp.

5.8.2.2 array_add_const()

```
void array_add_const (
            float * arr,
            float constant,
            size_t len )
```

Function to add a scalar constant to each element of an array.

Parameters

arr	Pointer to the array
constant	The constant scalar adder
len	Length of the array

Definition at line 96 of file array_utilities.cpp.

5.8.2.3 array_dot_product()

```
float array_dot_product (
            const float * in1,
            const float * in2,
            size_t len )
```

Function to calculate the dot-product of two 1-D vectors/arrays.

Parameters

in1	Pointer to the first vector
in2	Pointer to the second vector
len	Length of the vectors

Returns

Dot product (inner product) of the two vectors

Warning

Assumes both the vectors are of same length and takes only one length parameter

Definition at line 67 of file array_utilities.cpp.

5.8.2.4 array_element_divide_array()

```
void array_element_divide_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise division of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 132 of file array_utilities.cpp.

5.8.2.5 array_element_multiply_array()

```
void array_element_multiply_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise multiplication of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 123 of file array_utilities.cpp.

5.8.2.6 array_flip()

```
void array_flip (
            float * arr,
            size_t len )
```

Function to reverse an array.

Parameters

arr	Pointer to the array
len	Length of the array

Definition at line 31 of file array_utilities.cpp.

5.8.2.7 array_mean()

```
float array_mean (
            float * arr,
            size_t len )
```

Function to calculate the mean of the elements of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

Mean of the array elements

Definition at line 156 of file array_utilities.cpp.

5.8.2.8 array_mean_square()

```
float array_mean_square (
            const float * arr,
            size_t len )
```

Function to calculate the mean square of the elements of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

Mean square of the array elements

Definition at line 172 of file array_utilities.cpp.

5.8.2.9 array_min()

```
float array_min (
            const float * arr,
            size_t len )
```

Function to return the minimum of the elements of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

Minimum of the array elements

Definition at line 141 of file array_utilities.cpp.

5.8.2.10 array_multiply_const()

```
void array_multiply_const (
            float * arr,
            float constant,
            size_t len )
```

Function to multiply each element of an array by a scalar constant.

Parameters

arr	Pointer to the array
constant	The constant scalar multiplier
len	Length of the array

Definition at line 87 of file array_utilities.cpp.

5.8.2.11 array_print()

```
void array_print (
            const char * str,
            float * arr,
            size_t len )
```

Function to print an array for debugging.

Parameters

str	String to use for debugging
arr	Pointer to the array
len	Length of the array

Definition at line 177 of file array_utilities.cpp.

5.8.2.12 array_right_shift()

```
void array_right_shift (
            float * arr,
            size_t len )
```

Function to right shift an array by one place. Left most value will be replaced by zero.

Parameters

arr	Pointer to the array
len	Length of the array

Definition at line 78 of file array_utilities.cpp.

5.8.2.13 array_square()

```
void array_square (
            const float * in,
            float * out,
            size_t len )
```

Function to populate the output array with square of the elements of an input array.

Parameters

in	Pointer to the input array
out	Pointer to the output array
len	Length of the arrays

Warning

Assumes that output array already has memory allocated to it

Definition at line 163 of file array_utilities.cpp.

5.8.2.14 array_subtract_array()

```
void array_subtract_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise subtraction of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 114 of file array_utilities.cpp.

5.8.2.15 array_sum()

```
float array_sum (
            const float * arr,
            size_t len )
```

Function to calculate the sum of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

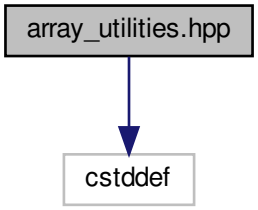
Sum of the array

Definition at line 47 of file array_utilities.cpp.

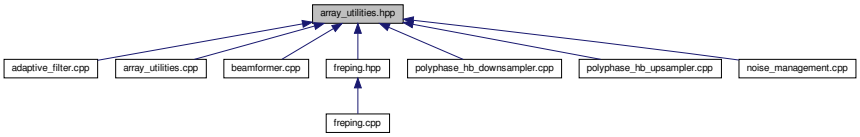
5.9 array_utilities.hpp File Reference

```
#include <cstddef>
```

Include dependency graph for array_utilities.hpp:



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



Functions

- void array_flip (float *arr, size_t len)

  Function to reverse an array.
- float array_sum (const float *arr, size_t len)

Function to calculate the sum of an array.

- float array_dot_product (const float *in1, const float *in2, size_t len)

  Function to calculate the dot-product of two 1-D vectors/arrays.
- void array_right_shift (float *arr, size_t len)

  Function to right shift an array by one place. Left most value will be replaced by zero.
- void array_multiply_const (float *arr, float constant, size_t len)

  Function to multiply each element of an array by a scalar constant.
- void array_add_const (float *arr, float constant, size_t len)

  Function to add a scalar constant to each element of an array.
- void array_add_array (float *in1, const float *in2, size_t len)

  Function to do element wise addition of two arrays.
- void array_subtract_array (float *in1, const float *in2, size_t len)

  Function to do element wise subtraction of two arrays.
- void array_element_multiply_array (float *in1, const float *in2, size_t len)

  Function to do element wise multiplication of two arrays.
- void array_element_divide_array (float *in1, const float *in2, size_t len)

  Function to do element wise division of two arrays.
- float array_min (const float *arr, size_t len)

  Function to return the minimum of the elements of an array.
- float array_mean (float *arr, size_t len)

  Function to calculate the mean of the elements of an array.
- void array_square (const float *in, float *out, size_t len)

  Function to populate the output array with square of the elements of an input array.
- float array_mean_square (const float *arr, size_t len)

  Function to calculate the mean square of the elements of an array.
- void array_print (const char *str, float *arr, size_t len)

  Function to print an array for debugging.

5.9.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.9.2 Function Documentation

5.9.2.1 array_add_array()

```
void array_add_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise addition of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 105 of file array_utilities.cpp.

5.9.2.2 array_add_const()

```
void array_add_const (
            float * arr,
            float constant,
            size_t len )
```

Function to add a scalar constant to each element of an array.

Parameters

arr	Pointer to the array
constant	The constant scalar adder
len	Length of the array

Definition at line 96 of file array_utilities.cpp.

5.9.2.3 array_dot_product()

```
float array_dot_product (
            const float * in1,
            const float * in2,
            size_t len )
```

Function to calculate the dot-product of two 1-D vectors/arrays.

Parameters

in1	Pointer to the first vector
in2	Pointer to the second vector
len	Length of the vectors

Returns

Dot product (inner product) of the two vectors

Warning

Assumes both the vectors are of same length and takes only one length parameter

Definition at line 67 of file array_utilities.cpp.

5.9.2.4 array_element_divide_array()

```
void array_element_divide_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise division of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 132 of file array_utilities.cpp.

5.9.2.5 array_element_multiply_array()

```
void array_element_multiply_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise multiplication of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 123 of file array_utilities.cpp.

5.9.2.6 array_flip()

```
void array_flip (
            float * arr,
            size_t len )
```

Function to reverse an array.

Parameters

arr	Pointer to the array
len	Length of the array

Definition at line 31 of file array_utilities.cpp.

5.9.2.7 array_mean()

```
float array_mean (
            float * arr,
            size_t len )
```

Function to calculate the mean of the elements of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

Mean of the array elements

Definition at line 156 of file array_utilities.cpp.

5.9.2.8 array_mean_square()

```
float array_mean_square (
            const float * arr,
            size_t len )
```

Function to calculate the mean square of the elements of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

Mean square of the array elements

Definition at line 172 of file array_utilities.cpp.

5.9.2.9 array_min()

```
float array_min (
            const float * arr,
            size_t len )
```

Function to return the minimum of the elements of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

Minimum of the array elements

Definition at line 141 of file array_utilities.cpp.

5.9.2.10 array_multiply_const()

```
void array_multiply_const (
            float * arr,
            float constant,
            size_t len )
```

Function to multiply each element of an array by a scalar constant.

Parameters

arr	Pointer to the array
constant	The constant scalar multiplier
len	Length of the array

Definition at line 87 of file array_utilities.cpp.

5.9.2.11 array_print()

```
void array_print (
            const char * str,
            float * arr,
            size_t len )
```

Function to print an array for debugging.

Parameters

str	String to use for debugging
arr	Pointer to the array
len	Length of the array

Definition at line 177 of file array_utilities.cpp.

5.9.2.12 array_right_shift()

```
void array_right_shift (
            float * arr,
            size_t len )
```

Function to right shift an array by one place. Left most value will be replaced by zero.

Parameters

arr	Pointer to the array
len	Length of the array

Definition at line 78 of file array_utilities.cpp.

5.9.2.13 array_square()

```
void array_square (
            const float * in,
            float * out,
            size_t len )
```

Function to populate the output array with square of the elements of an input array.

Parameters

in	Pointer to the input array
out	Pointer to the output array
len	Length of the arrays

Warning

Assumes that output array already has memory allocated to it

Definition at line 163 of file array_utilities.cpp.

5.9.2.14 array_subtract_array()

```
void array_subtract_array (
            float * in1,
            const float * in2,
            size_t len )
```

Function to do element wise subtraction of two arrays.

Parameters

in1	Pointer to the first array
in2	Pointer to the second array
len	Length of the arrays

Warning

Assumes both the arrays are of same length and takes only one length parameter

Definition at line 114 of file array_utilities.cpp.

5.9.2.15 array_sum()

```
float array_sum (
            const float * arr,
            size_t len )
```

Function to calculate the sum of an array.

Parameters

arr	Pointer to the array
len	Length of the array

Returns

Sum of the array

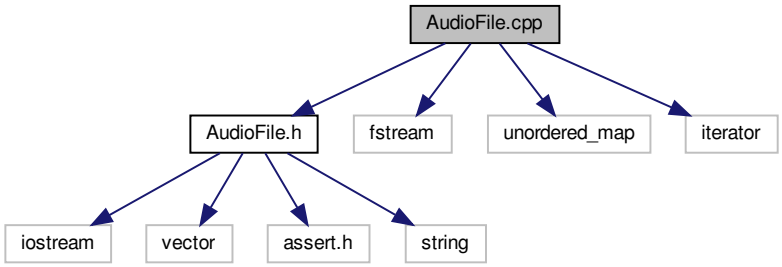
Definition at line 47 of file array_utilities.cpp.

5.10 AudioFile.cpp File Reference

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

```
#include <unordered_map>
#include <iterator>
```

Include dependency graph for AudioFile.cpp:



Variables

- std::unordered_map< uint32_t, std::vector< uint8_t > > **aiffSampleRateTable**

5.10.1 Detailed Description

Author

Adam Stark

Copyright

This file is part of the 'AudioFile' library

5.10.2 Variable Documentation

5.10.2.1 aiffSampleRateTable

```
std::unordered_map<uint32_t, std::vector<uint8_t> > aiffSampleRateTable
```

Initial value:

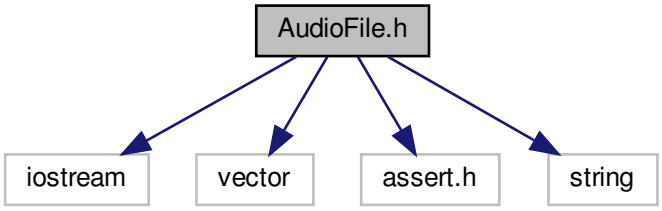
```
= {
    {8000, {64, 11, 250, 0, 0, 0, 0, 0, 0, 0}},
    {11025, {64, 12, 172, 68, 0, 0, 0, 0, 0, 0}},
    {16000, {64, 12, 250, 0, 0, 0, 0, 0, 0, 0}},
    {22050, {64, 13, 172, 68, 0, 0, 0, 0, 0, 0}},
    {32000, {64, 13, 250, 0, 0, 0, 0, 0, 0, 0}},
    {37800, {64, 14, 147, 168, 0, 0, 0, 0, 0, 0}},
    {44056, {64, 14, 172, 24, 0, 0, 0, 0, 0, 0}},
    {44100, {64, 14, 172, 68, 0, 0, 0, 0, 0, 0}},
    {47250, {64, 14, 184, 146, 0, 0, 0, 0, 0, 0}},
    {48000, {64, 14, 187, 128, 0, 0, 0, 0, 0, 0}},
    {50000, {64, 14, 195, 80, 0, 0, 0, 0, 0, 0}},
    {50400, {64, 14, 196, 224, 0, 0, 0, 0, 0, 0}},
    {88200, {64, 15, 172, 68, 0, 0, 0, 0, 0, 0}},
    {96000, {64, 15, 187, 128, 0, 0, 0, 0, 0, 0}},
    {176400, {64, 16, 172, 68, 0, 0, 0, 0, 0, 0}},
    {192000, {64, 16, 187, 128, 0, 0, 0, 0, 0, 0}},
    {352800, {64, 17, 172, 68, 0, 0, 0, 0, 0, 0}},
    {2822400, {64, 20, 172, 68, 0, 0, 0, 0, 0, 0}},
    {5644800, {64, 21, 172, 68, 0, 0, 0, 0, 0, 0}}
}
```

Definition at line 30 of file AudioFile.cpp.

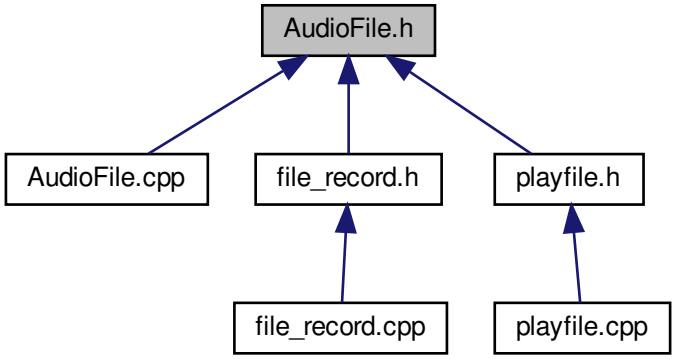
5.11 AudioFile.h File Reference

```
#include <iostream>
#include <vector>
#include <assert.h>
#include <string>
```

Include dependency graph for AudioFile.h:



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



Classes

- class AudioFile< T >

Enumerations

- enum AudioFileFormat { **Error**, **NotLoaded**, **Wave**, **Aiff** }

5.11.1 Detailed Description

Author

Adam Stark

Copyright

Copyright (C) 2017 Adam Stark

This file is part of the 'AudioFile' library

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

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see http://www.gnu.org/licenses/.

5.11.2 Enumeration Type Documentation

5.11.2.1 AudioFileFormat

```
enum AudioFileFormat  [strong]
```

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

Definition at line 37 of file AudioFile.h.

5.12 bandlimited_filter.h File Reference

Macros

- #define **BANDLIMITED_FILTER_SIZE** 3

Variables

- float **bandlimited_filter** [BANDLIMITED_FILTER_SIZE]

5.12.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.12.2 Variable Documentation

5.12.2.1 bandlimited_filter

```
float bandlimited_filter[BANDLIMITED_FILTER_SIZE]
```

Initial value:

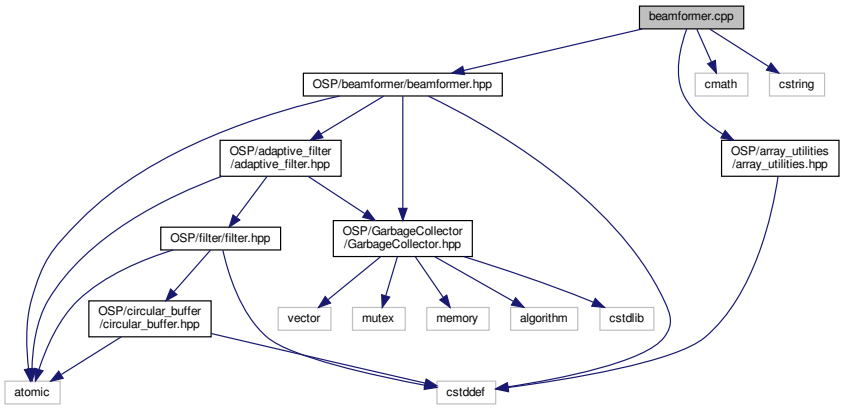
```
= {
        1.0f,
        -1.8f,
        0.81f
}
```

Definition at line 33 of file bandlimited_filter.h.

5.13 beamformer.cpp File Reference

```
#include <OSP/beamformer/beamformer.hpp>
#include <OSP/array_utilities/array_utilities.hpp>
#include <cmath>
#include <cstring>
```

Include dependency graph for beamformer.cpp:



5.13.1 Detailed Description

Author

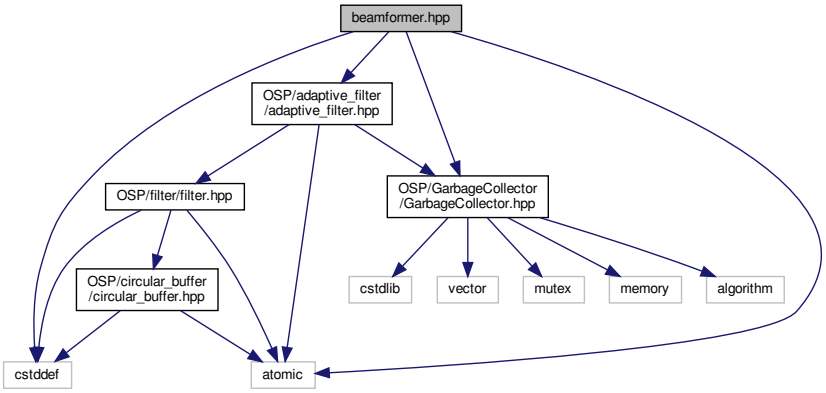
Open Speech Platform (OSP) Team, UCSD

Copyright

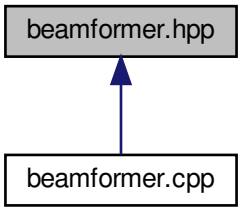
5.14 beamformer.hpp File Reference

```
#include <cstddef>
#include <OSP/adaptive_filter/adaptive_filter.hpp>
#include <atomic>
#include <OSP/GarbageCollector/GarbageCollector.hpp>
```

Include dependency graph for beamformer.hpp:



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



Classes

- class beamformer

  Beamformer Class.

5.14.1 Detailed Description

Author

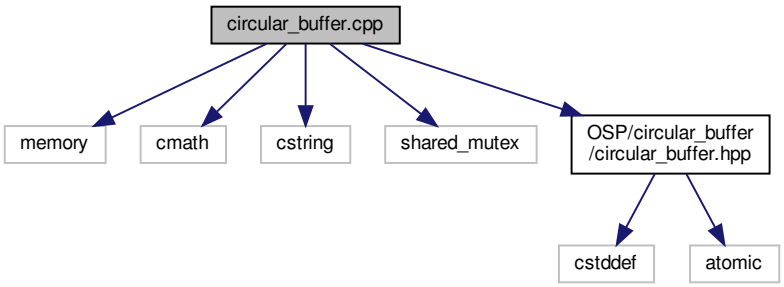
Open Speech Platform (OSP) Team, UCSD

Copyright

5.15 circular_buffer.cpp File Reference

```
#include <memory>
#include <cmath>
#include <cstring>
#include <shared_mutex>
#include <OSP/circular_buffer/circular_buffer.hpp>
```

Include dependency graph for circular_buffer.cpp:



5.15.1 Detailed Description

Author

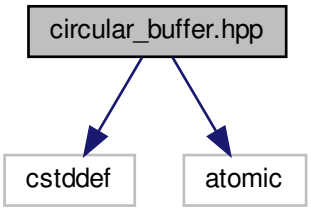
Open Speech Platform (OSP) Team, UCSD

Copyright

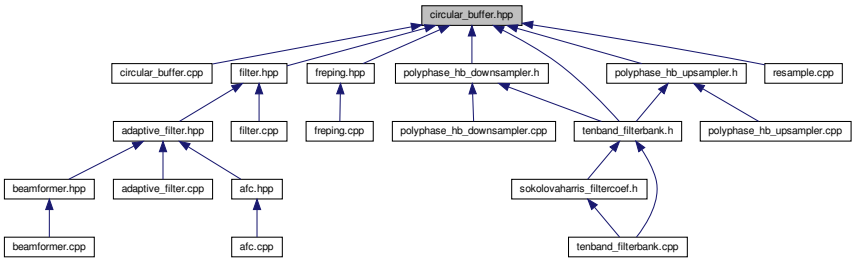
5.16 circular_buffer.hpp File Reference

```
#include <cstddef>
#include <atomic>
```

Include dependency graph for circular_buffer.hpp:



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



Classes

- class circular_buffer

  Circular Buffer Class.

5.16.1 Detailed Description

Author

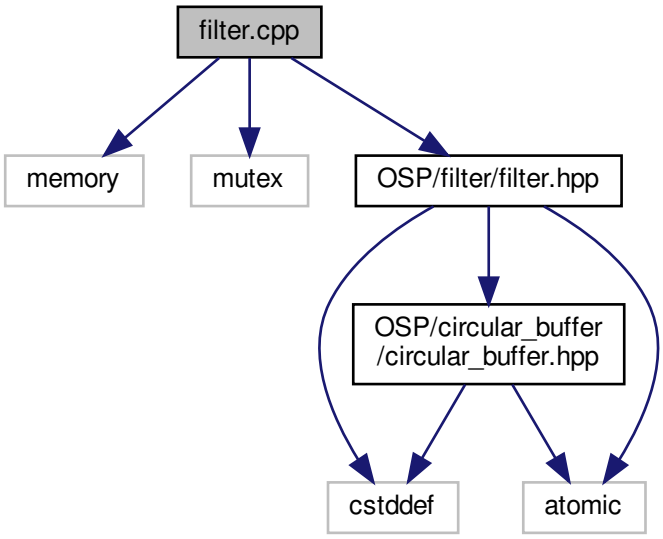
Open Speech Platform (OSP) Team, UCSD

Copyright

5.17 filter.cpp File Reference

```
#include <memory>
#include <mutex>
#include <OSP/filter/filter.hpp>
```

Include dependency graph for filter.cpp:



5.17.1 Detailed Description

Author

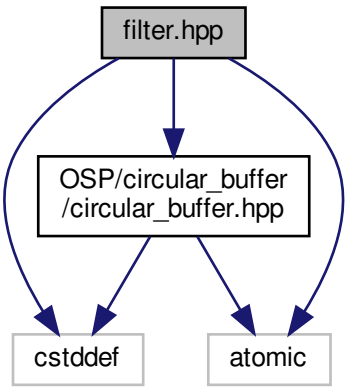
Open Speech Platform (OSP) Team, UCSD

Copyright

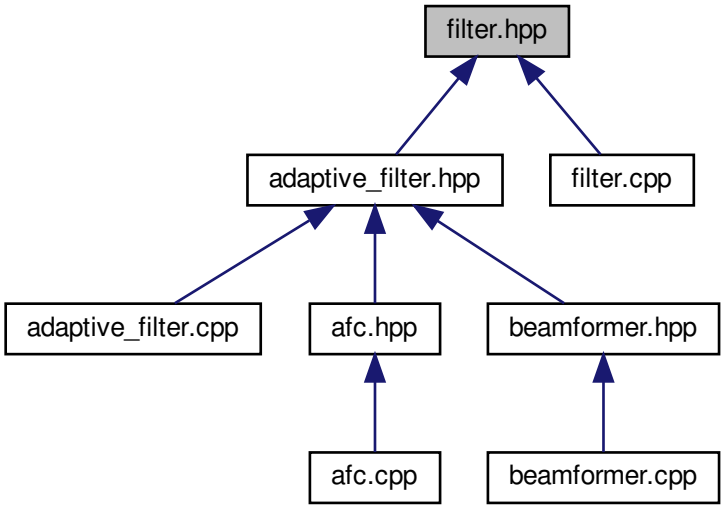
5.18 filter.hpp File Reference

```
#include <cstddef>
#include <atomic>
#include <OSP/circular_buffer/circular_buffer.hpp>
```

Include dependency graph for filter.hpp:



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



Classes

- class filter

  Filter Class.

5.18.1 Detailed Description

Author

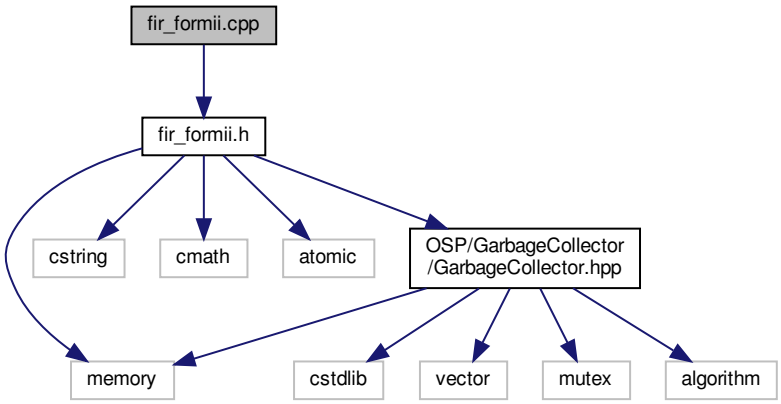
Open Speech Platform (OSP) Team, UCSD

Copyright

5.19 fir_formii.cpp File Reference

```
#include "fir_formii.h"
```

Include dependency graph for fir_formii.cpp:



5.19.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

Copyright (C) 2020 Regents of the University of California Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

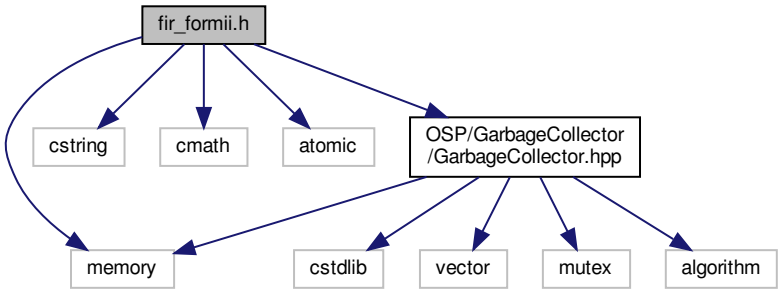
```
1. Redistributions of source code must retain the above copyright notice, this list of conditions and the
following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the
following disclaimer in the documentation and/or other materials provided with the distribution.
```

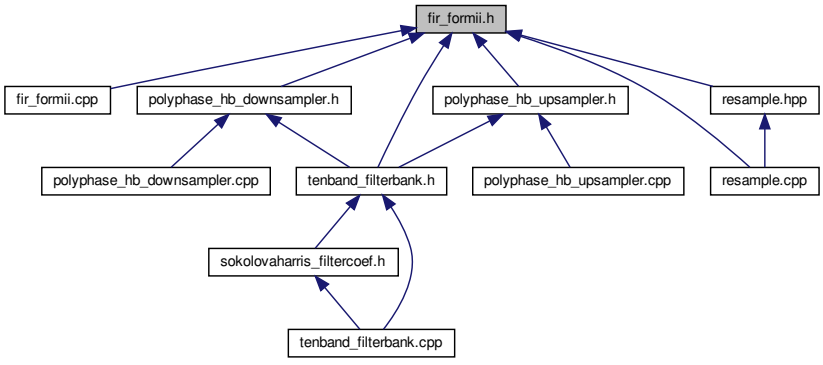
5.20 fir_formii.h File Reference

```
#include <memory>
#include <cstring>
#include <cmath>
#include <atomic>
#include <OSP/GarbageCollector/GarbageCollector.hpp>
```

Include dependency graph for fir_formii.h:



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



Classes

- class fir_formii

  Filter Class.

5.20.1 Detailed Description

Author

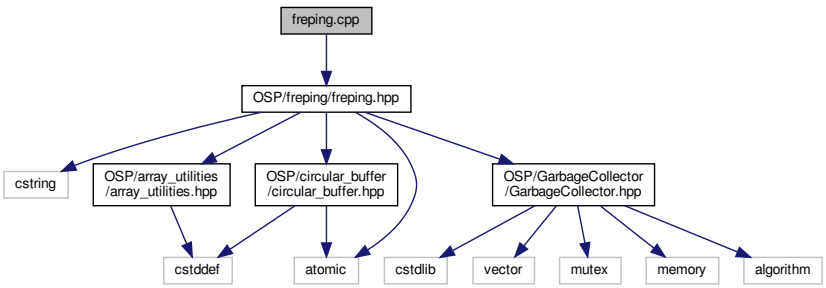
Open Speech Platform (OSP) Team, UCSD

Copyright

5.21 freping.cpp File Reference

```
#include <OSP/freping/freping.hpp>
```

Include dependency graph for freping.cpp:



5.21.1 Detailed Description

Author

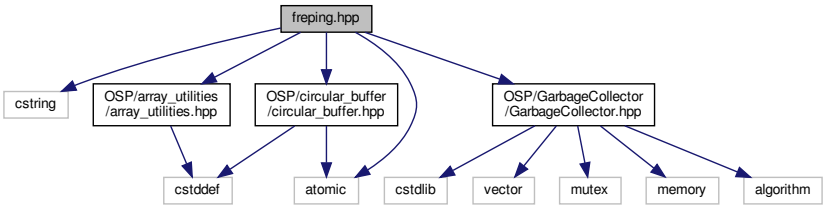
Open Speech Platform (OSP) Team, UCSD

Copyright

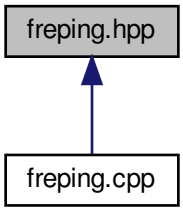
5.22 freping.hpp File Reference

```
#include <cstring>
#include <OSP/circular_buffer/circular_buffer.hpp>
#include <OSP/array_utilities/array_utilities.hpp>
#include <atomic>
#include <OSP/GarbageCollector/GarbageCollector.hpp>
```

Include dependency graph for freping.hpp:



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



Classes

- class freping

  Freping Class.

5.22.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.23 hamming_window128.h File Reference

Variables

- long **hamming_window128_length** = 128

5.23.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.24 hamming_window64.h File Reference

Variables

- long **hamming_window64_length** = 64

5.24.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.25 prefilter.h File Reference

Macros

- #define **PREFILTER_SIZE** 3

Variables

- float **prefilter** [PREFILTER_SIZE]

5.25.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright

5.25.2 Variable Documentation

5.25.2.1 prefilter

```
float prefilter[PREFILTER_SIZE]
```

Initial value:

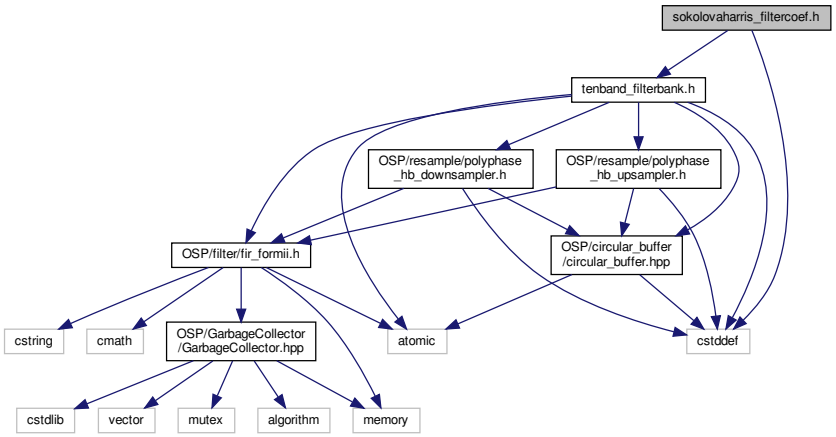
```
= {
        1.0f,
        -2.01f,
        1.0f
}
```

Definition at line 32 of file prefilter.h.

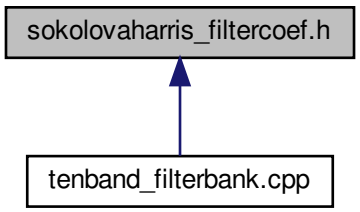
5.26 sokolovaharris_filtercoef.h File Reference

```
#include <cstddef>
#include "tenband_filterbank.h"
```

Include dependency graph for sokolovaharris_filtercoef.h:



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



Variables

- size_t **half_band_filter1_length** = 35
- size_t **half_band_filter2_length** = 11
- size_t **half_band_filter3_length** = 7

5.26.1 Detailed Description

Author

Open Speech Platform (OSP) Team, UCSD

Copyright