Ambiente de Teste para Filtros Adaptativos

Generated by Doxygen 1.8.6

Sun Mar 9 2014 17:00:44

Contents

Chapter 1

Main Page

Projeto Final de Graduação

Para compilar:

```
                [  pf/  ]$ cd build
          [  pf/build/  ]$ ./config.sh
          [  pf/build/  ]$ cd release
 [  pf/build/release/  ]$ make
```

O executável será colocado no diretório `build/release`.

Para gerar um executável do *Debug Build*, basta substituir o `cd release` por `cd debug`. O novo diretório do executável será `build/debug`.

Manual em PDF `aqui`.

Chapter 2

Todo List

Member Signal::Signal (const std::string &filename)

add "extern C" directive (speed?)

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

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

Chapter 4

Class Index

4.1 Class List

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

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

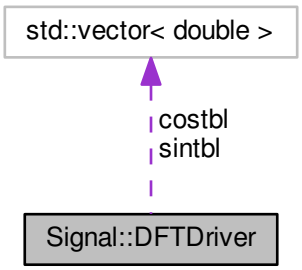
Class Documentation

6.1 Signal::DFTDriver Class Reference

A class for providing discrete Fourier transform capabilities.

`#include <Signal.h>`

Collaboration diagram for Signal::DFTDriver:



Public Types

- enum dir_t { DIRECT, INVERSE }

Public Member Functions

- DFTDriver ()

  Constructor for an object that computes DFTs.
- void operator() (container_t &re, container_t &im, dir_t direction=DIRECT)

  Used to perform the actual computation of the DFT.

Static Public Member Functions

- static std::vector< double > initialize_costbl ()

Initializes the table of cosines.

- static std::vector< double > initialize_sintbl ()

  Initializes the table of sines.

Static Public Attributes

- static const unsigned tblbits = 14

  Number of bits for the index of the table of sines and cosines.

- static const size_t tblsize = 16384

  Number of entries in the tables of sines and cosines.

Private Member Functions

- double Wre (unsigned k)

  Easy access to the table of cosines.

- double Wim (unsigned k)

  Easy access to the table of sines.

Static Private Member Functions

- template<typename T >
  static T br (T x, int bits)

  Bit-reverse.

Private Attributes

- unsigned bits

  Number of bits for the current FFT computation.

Static Private Attributes

- static const std::vector< double > sintbl

  Table of sines.

- static const std::vector< double > costbl

  Table of cosines.

6.1.1 Detailed Description

A class for providing discrete Fourier transform capabilities.

This class implements the radix-2 FFT algorithm used in the Signal::filter() method.

Usage:

```
Signal::DFTDriver dft;
Signal::container_t real, imag;
// initialize the real and imaginary parts of a complex time-domain
// signal
dft(real, imag); // performs in-place FFT
// now, work with the real and imaginary parts of the
// frequency-domain version of the signal
dft(real, imag, Signal::DFTDriver::INVERSE); // inverse in-place fft
// now, we can work again with the time-domain complex signal
```

Definition at line 191 of file Signal.h.

6.1.2 Member Enumeration Documentation

6.1.2.1 enum Signal::DFTDriver::dir_t

This is a type for specifying whether we should perform a direct or inverse FFT.

Enumerator

DIRECT Perform direct FFT.

INVERSE Perform inverse FFT.

Definition at line 336 of file Signal.h.

6.1.3 Constructor & Destructor Documentation

6.1.3.1 Signal::DFTDriver::DFTDriver ( ) [inline]

Constructor for an object that computes DFTs.

Does nothing at all.

Definition at line 372 of file Signal.h.

6.1.4 Member Function Documentation

6.1.4.1 template<typename T > static T Signal::DFTDriver::br (T *x*, int *bits*) [inline], [static], [private]

Bit-reverse.

Returns the bit-reversed version of the parameter *x*. Assumes *x* is *bits*-bit wide, and ignore any bits with more significance than that.

This function assumes that the number of bits in one `char` is 8, and that bitshifting is zero-padded, and not circular.

Template Parameters

T	The type of the parameter *x*. It **must** be an unsigned integer type.

Parameters

in	*x*	The *bits*-bit unsigned integer to be bit-reversed.
in	*bits*	The number of bits of the integer *x*.

Returns

the unsigned integer *x*, bit-reversed.

Definition at line 217 of file Signal.h.

Referenced by operator()().

6.1.4.2 static std::vector<double> Signal::DFTDriver::initialize_costbl ( ) [inline], [static]

Initializes the table of cosines.

Computes a table of cosines that will be handed to the `costbl` member.

See Also

initialize_sintbl
costbl

Definition at line 349 of file Signal.h.

References TAU, and tblsize.

6.1.4.3 static std::vector<double> Signal::DFTDriver::initialize_sintbl ( ) [inline], [static]

Initializes the table of sines.

See Also

initialize_costbl
sintbl

Definition at line 361 of file Signal.h.

References TAU, and tblsize.

6.1.4.4 void Signal::DFTDriver::operator() (container_t & *re*, container_t & *im*, dir_t *direction* = DIRECT)

Used to perform the actual computation of the DFT.

Implements the radix-2 time-decimation FFT algorithm. The computation happens in-place, which means that the *re* and *im* parameters are substituted by their new versions.

Of course, the *re* and *im* vectors must be of the same size. This size must be a power of two not greater than tblsize.

Refer to the DFTDriver class documentation for usage details.

Exceptions

std::runtime_error	if any of the above conditions aren't met.

Parameters

in, out	*re*	Real part of the compelx signal on which the FFT will act.
in, out	*im*	Imaginary part.
in	*direction*	Wether this is a direct or inverse DFT.

Definition at line 446 of file Signal.cpp.

References bits, br(), DIRECT, INVERSE, tblsize, Wim(), and Wre().

6.1.4.5 double Signal::DFTDriver::Wim (unsigned *k*) [inline], [private]

Easy access to the table of sines.

Parameters

in	*k*	Same as in Wre.

Returns

$$\sin\left(\tau \cdot \mathrm{k}/2^{\mathrm{bits}}\right)$$

See Also

sintbl
Wre

Definition at line 310 of file Signal.h.

References bits, sintbl, and tblbits.

Referenced by operator()().

6.1.4.6 double Signal::DFTDriver::Wre (unsigned *k*) [inline], [private]

Easy access to the table of cosines.

This function is aware of the number of bits of the current FFT, and makes it easy to get the cosine of $\tau \cdot \mathrm{k}/2^{\mathrm{bits}}$, using the pre-computed table of cosines.

Parameters

in	*k*	An integer in the range	$0, 2^{\mathrm{bits}}$.

Returns

$\cos\left(\tau \cdot \mathrm{k}/2^{\mathrm{bits}}\right)$, where τ is shorthand for 2π.

See Also

costbl
Wim

Definition at line 298 of file Signal.h.

References bits, costbl, and tblbits.

Referenced by operator()().

6.1.5 Member Data Documentation

6.1.5.1 unsigned Signal::DFTDriver::bits [private]

Number of bits for the current FFT computation.

Always assume this is uninitialized, and all methods that use it should initialize it themselves.

Definition at line 198 of file Signal.h.

Referenced by operator()(), Wim(), and Wre().

6.1.5.2 const std::vector< double > Signal::DFTDriver::costbl [static], [private]

Initial value:

```
=
        Signal::DFTDriver::initialize_costbl()
```

Table of cosines.

See Also

sintbl

Definition at line 393 of file Signal.h.

Referenced by Wre().

6.1.5.3 const std::vector< double > Signal::DFTDriver::sintbl [static], [private]

Initial value:

```
=
        Signal::DFTDriver::initialize_sintbl()
```

Table of sines.

Holds the sines of $\tau \cdot k/\mathtt{tblsize}$, for k in the range $[0, \mathtt{tblsize}[$. Here, τ is shorthand for 2π.

See Also

costbl

Definition at line 387 of file Signal.h.

Referenced by Wim().

6.1.5.4 const unsigned Signal::DFTDriver::tblbits = 14 [static]

Number of bits for the index of the table of sines and cosines.

We won't be able to perform an N-bit dft if $N > \mathtt{tblbits}$, so this should be big. Also, this **must** be equal to $\log_2(\mathtt{tblsize})$, but there's nothing in the source code that enforces it.

See Also

tblsize

Definition at line 323 of file Signal.h.

Referenced by Wim(), and Wre().

6.1.5.5 const size_t Signal::DFTDriver::tblsize = 16384 [static]

Number of entries in the tables of sines and cosines.

This **must** be equal to $2^{\mathtt{tblbits}}$, but there's nothing in the source code that enforces it.

See Also

tblbits

Definition at line 332 of file Signal.h.

Referenced by Signal::filter(), initialize_costbl(), initialize_sintbl(), and operator()().

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

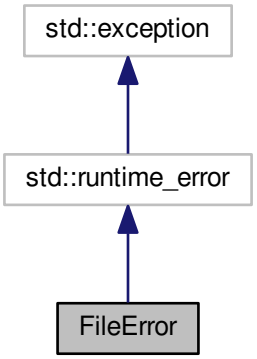
- Signal.h (v0.1.1-17-gc9cc97d)
- Signal.cpp (v0.1.1-17-gc9cc97d)

6.2 FileError Class Reference

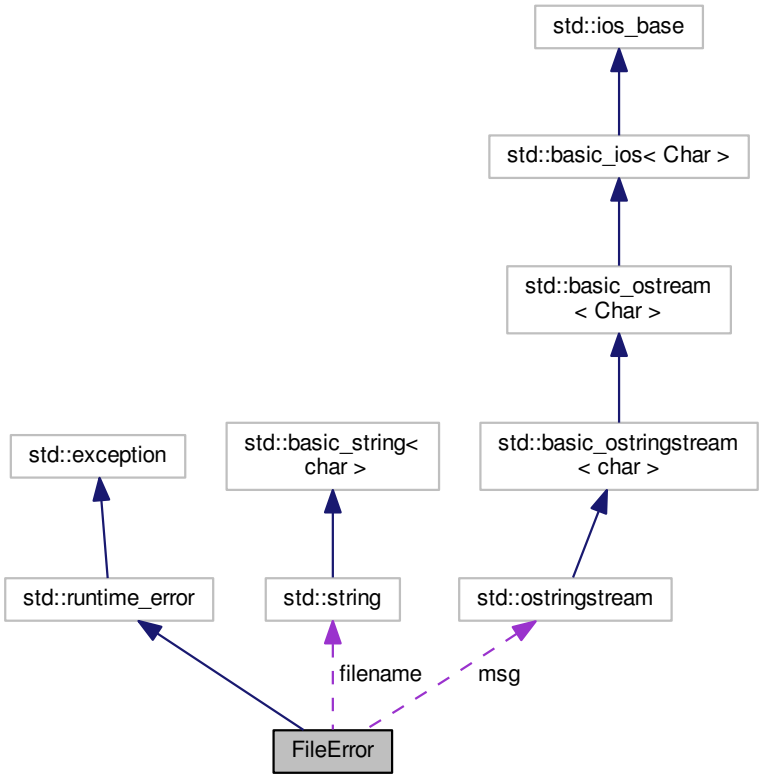
A runtime exception while trying to process a file.

`#include <utils.h>`

Inheritance diagram for FileError:



Collaboration diagram for FileError:



Public Member Functions

- FileError (const std::string &fn)

  Constructs the exception object from the filename.
- ~FileError () throw ()

  Destructor that does nothing.
- virtual const char * what () const throw ()

  Gives a description for the error.

Private Attributes

- const std::string filename

  The name of the file that caused the error.

Static Private Attributes

- static std::ostringstream msg

  The message that will be displayed if we don't catch the exception.

6.2.1 Detailed Description

A runtime exception while trying to process a file.

Thrown when we cannot read a file, for some reason.

Usage:

```
if (error ocurred) throw FileError("badfile.wav");
```

Or:

```
std::string filename;
std::cin >> filename;
// ...
if (error ocurred) throw FileError(filename);
```

Definition at line 77 of file utils.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 FileError::FileError (const std::string & *fn*) [inline]

Constructs the exception object from the filename.

Parameters

in	*fn*	A `std::string` that holds the filename.

Definition at line 95 of file utils.h.

6.2.2.2 FileError::~FileError ( ) throw) [inline]

Destructor that does nothing.

Needed to prevent the `looser throw specifier` error because, `std::runtime_error::~runtime_error()` is declared as `throw()`

Definition at line 103 of file utils.h.

6.2.3 Member Function Documentation

6.2.3.1 virtual const char* FileError::what ( ) const throw) [inline], [virtual]

Gives a description for the error.

Updates the msg static member with the error message, and returns it as a C string.

Definition at line 110 of file utils.h.

References filename, and msg.

6.2.4 Member Data Documentation

6.2.4.1 const std::string FileError::filename [private]

The name of the file that caused the error.

Definition at line 88 of file utils.h.

Referenced by what().

6.2.4.2 std::ostringstream FileError::msg [static], [private]

The message that will be displayed if we don't catch the exception.

Must be static, so that we can modify it inside the `what() const` function, and read it after the temporary object has been destroyed.

Definition at line 85 of file utils.h.

Referenced by what().

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

- utils.h (v0.1.1-17-gc9cc97d)
- utils.cpp (v0.1.1-17-gc9cc97d)

6.3 Stream::sample_wrapper_t Struct Reference

A structured type for holding a single sample. Will be simplified later.

`#include <Stream.h>`

Public Member Functions

- sample_wrapper_t ()
- sample_wrapper_t (sample_t s)

Public Attributes

- sample_t sample

6.3.1 Detailed Description

A structured type for holding a single sample. Will be simplified later.

Definition at line 75 of file Stream.h.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Stream::sample_wrapper_t::sample_wrapper_t ( ) [inline]

Definition at line 78 of file Stream.h.

6.3.2.2 Stream::sample_wrapper_t::sample_wrapper_t (sample_t *s*) [inline]

Definition at line 79 of file Stream.h.

6.3.3 Member Data Documentation

6.3.3.1 sample_t Stream::sample_wrapper_t::sample

Definition at line 77 of file Stream.h.

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

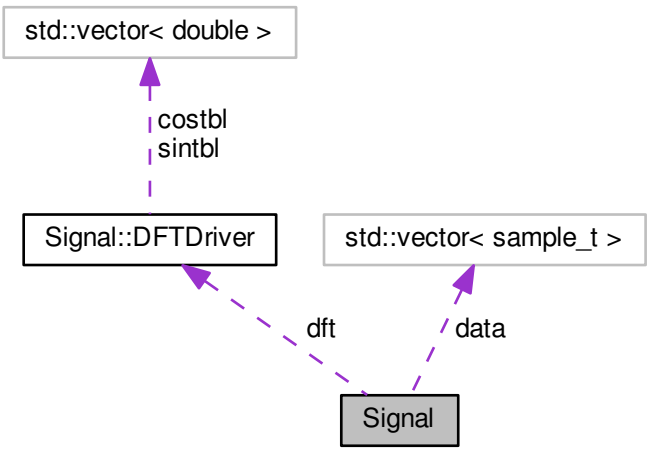
- Stream.h (v0.1.1-17-gc9cc97d)

6.4 Signal Class Reference

A time- or frequency-domain signal.

`#include <Signal.h>`

Collaboration diagram for Signal:



Classes

- class DFTDriver

  A class for providing discrete Fourier transform capabilities.

Public Types

- enum delay_t { MS, SAMPLE }

  This is a type for specifying whether a time interval is given in milliseconds or in samples.
- typedef float sample_t

  The type for holding each signal sample.
- typedef unsigned long index_t

  The type for holding each signal sample index.
- typedef std::vector< sample_t > container_t

  The type for holding the whole vector of signal samples.

Public Member Functions

- Signal ()

  Constructs an empty signal.
- Signal (const std::string &filename)

  Constructs a signal from an audio file.
- Signal (const Signal &other)

  Copy-constructor. Constructs a signal as a copy of another.

- ~Signal ()

  Frees memory used.
- const sample_t * array () const

  Returns a pointer to the first sample.
- index_t samples () const

  Number of samples.
- int samplerate () const

  Sample rate in samples per second.
- sample_t & operator[] (index_t index)

  Returns a sample.
- const sample_t & operator[] (index_t index) const

  Returns a "read-only" sample.
- void set_size (index_t n)

  Changes the number of samples.
- void set_samplerate (int sr)

  Re-samples the signal.
- void delay (delay_t t, unsigned long d)

  Delays the signal in time.
- void gain (double g)

  Applies gain g to the signal.
- sample_t l_inf_norm ()

  Gets the ℓ^∞-norm of the signal.
- void normalize ()

  Normalize the signal according to its ℓ^∞-norm.
- Signal & operator+= (Signal other)

  Adds the other signal to the caller.
- void filter (Signal imp_resp)

  Convolves the sinal with an impulse response.
- void play (bool sleep=true)

  Makes PortAudio playback the audio signal.

Public Attributes

- index_t counter

  general-purpose variable for external use.

Static Public Attributes

- static DFTDriver dft

  Single instance of the DFTDriver class.

Private Attributes

- container_t data

  Holds the signal samples.
- int srate

  Signal sample rate in Hertz.

6.4.1 Detailed Description

A time- or frequency-domain signal.

Holds data and provides routines for dealing with time-domain and frequency-domain signals. Currently, all Signals are an array of single-precision floating-point samples. Signals are aware of their sample rates.

Definition at line 35 of file Signal.h.

6.4.2 Member Typedef Documentation

6.4.2.1 typedef std::vector<sample_t> Signal::container_t

The type for holding the whole vector of signal samples.

Definition at line 46 of file Signal.h.

6.4.2.2 typedef unsigned long Signal::index_t

The type for holding each signal sample index.

Definition at line 43 of file Signal.h.

6.4.2.3 typedef float Signal::sample_t

The type for holding each signal sample.

Definition at line 40 of file Signal.h.

6.4.3 Member Enumeration Documentation

6.4.3.1 enum Signal::delay_t

This is a type for specifying whether a time interval is given in milliseconds or in samples.

Enumerator

MS Time interval given in milliseconds.

SAMPLE Time interval given in samples.

Definition at line 50 of file Signal.h.

6.4.4 Constructor & Destructor Documentation

6.4.4.1 Signal::Signal ( ) [inline]

Constructs an empty signal.

Initializes the signal with no meta-data and no samples. The user needs to specify the sample rate and create samples before using the signal.

Definition at line 60 of file Signal.h.

6.4.4.2 Signal::Signal (const std::string & *filename*)

Constructs a signal from an audio file.

Constructs a signal getting the signal data from an audio file. This is done using the libsndfile library. The filetypes supported are listed here. WAV is supported, but MP3 is not.

If the given file is stereo, or otherwise multi-channel, just the first channel will be read. (On stereo audio files, this is the left channel.)

The sample rate is extracted from the file's meta-data info.

Parameters

in	*filename*	Audio file name.

Todo add "extern C" directive (speed?)

Exceptions

FileError	if file openening/reading fails.

Definition at line 62 of file Signal.cpp.

References data, samples(), set_size(), and srate.

6.4.4.3 Signal::Signal (const Signal & *other*) [inline]

Copy-constructor. Constructs a signal as a copy of another.

Constructs a signal as a copy of another one. If this signal is not empty, we destroy it.

Parameters

in	*other*	The signal to be copied from.

Definition at line 72 of file Signal.h.

6.4.4.4 Signal::~Signal ( ) [inline]

Frees memory used.

Free the pointer to the array of samples.

Definition at line 79 of file Signal.h.

6.4.5 Member Function Documentation

6.4.5.1 const sample_t* Signal::array ( ) const [inline]

Returns a pointer to the first sample.

Sometimes needed for performance reasons. Shouldn't be used to modify the samples.

Returns

a pointer to the first element of a contiguous region of memory that holds the samples.

Definition at line 89 of file Signal.h.

References data.

6.4.5.2 void Signal::delay (delay_t *t*, unsigned long *d*)

Delays the signal in time.

Adds zeroed samples at the beginning of the signal.

If we try do delay a signal by milliseconds, but the signal has no associated sample rate, a warning is emitted, and nothing is done. No exception is thrown.

Parameters

in	*t*	A delay type element.
in	*d*	The time interval to be delayed, given in the units specified by `t`.

Definition at line 105 of file Signal.cpp.

References data, MS, samples(), set_size(), and srate.

6.4.5.3 void Signal::filter (Signal *imp_resp*)

Convolves the sinal with an impulse response.

Convolves the signal with the given finite impulse response (FIR).

The algorthm used is the "overlap-and-add", and we use the FFT implemented in the DFTDriver class to compute each step. We try to do it using the least possible number of DFTs.

Parameters

in	*imp_resp*	The filter impulse response to be convolved with.

See Also

DFTDriver::operator()

Definition at line 132 of file Signal.cpp.

References data, dft, Signal::DFTDriver::INVERSE, samples(), set_samplerate(), set_size(), srate, and Signal::DFTDriver::tblsize.

6.4.5.4 void Signal::gain (double *g*)

Applies gain *g* to the signal.

This can be useful, for example, to make sure that the signal is within the $[-1, 1]$ range.

Parameters

in	*g*	The signal gain to be applied.

Definition at line 410 of file Signal.cpp.

References data.

Referenced by normalize().

6.4.5.5 Signal::sample_t Signal::l_inf_norm ( )

Gets the ℓ^∞-norm of the signal.

Take the signal's infinity-norm, which is the maximum absolute value of all the samples of the signal.

Returns

the ℓ^∞-norm of the signal.

Definition at line 422 of file Signal.cpp.

References data.

Referenced by normalize().

6.4.5.6 void Signal::normalize ( ) [inline]

Normalize the signal according to its ℓ^∞-norm.

Divide the signal by a constant so that the maximum absolute value of its samples is 1.

Definition at line 163 of file Signal.h.

References gain(), and l_inf_norm().

6.4.5.7 Signal & Signal::operator+= (Signal *other*)

Adds the *other* signal to the caller.

First, we re-sample *other* into a new temporary signal. Then we increase the caller's size if needed, and finally add the signals sample-by-sample.

Parameters

in	*other*	The signal to be added to the caller.

Returns

a reference to this signal, already added to the `other`.

Definition at line 395 of file Signal.cpp.

References data, samples(), set_samplerate(), set_size(), and srate.

6.4.5.8 sample_t& Signal::operator[] (index_t *index*) [inline]

Returns a sample.

Gets a sample of the signal. For performance reasons, this method does not check that the given index is valid. (Except in debug releases, in which it *does* check.)

Parameters

in	*index*	The index of the desired sample. Signal indexes are zero-based.

Returns

a reference to the sample.

Definition at line 117 of file Signal.h.

References data, and samples().

6.4.5.9 const sample_t& Signal::operator[] (index_t *index*) const [inline]

Returns a "read-only" sample.

Just like the "read-write" version, but returns a const reference to a sample.

Parameters

in	*index*	The index of the desired sample. Signal indexes are zero-based.

Returns

a const reference to the sample.

Definition at line 134 of file Signal.h.

References data, and samples().

6.4.5.10 void Signal::play (bool *sleep* = `true`)

Makes PortAudio playback the audio signal.

Creates a PortAudio session for audio playback of the signal content. If *sleep* is `true`, we wait for the playback to end before returning. (If it's false, the function returns, while playback goes on in the background.)

Parameters

in	*sleep*	If set to true, the method will only return when the playback ends (that is, when the end of the signal is reached). Otherwise, it returns imediatly, and the playback goes on in the background.

Exceptions

std::runtime_error	if any of the PortAudio steps fail (check the source code)

See Also

signal_callback

Definition at line 315 of file Signal.cpp.

References counter, samples(), signal_callback(), and srate.

6.4.5.11 int Signal::samplerate ( ) const [inline]

Sample rate in samples per second.

Returns

the number of samples per second that should be used when playing back the signal.

Definition at line 102 of file Signal.h.

References srate.

6.4.5.12 index_t Signal::samples ( ) const [inline]

Number of samples.

Returns

the number of elements inside the vector of samples.

Definition at line 95 of file Signal.h.

References data.

Referenced by delay(), filter(), operator+=(), operator[](), play(), set_samplerate(), and Signal().

6.4.5.13 void Signal::set_samplerate (int *sr*)

Re-samples the signal.

Changes the sample rate of the signal. The way it is done, this is equivalent to reconstructing the time-domain signal by linear interpolation, and then re-sampling the continuous-time reconstructed signal at the new sample rate.

Parameters

in	*sr*	The new sample rate in Hertz.

See Also

srate

Definition at line 372 of file Signal.cpp.

References data, samples(), and srate.

Referenced by filter(), and operator+=().

6.4.5.14 void Signal::set_size (index_t *n*) [inline]

Changes the number of samples.

Changes the signal length. Allocates more space if we are growing the signal, and deletes the last samples if we are shrinking it. Also initializes any new samples to zero.

Parameters

in	*n*	The desired signal length.

See Also

container_t::resize()

Definition at line 151 of file Signal.h.

References data.

Referenced by delay(), filter(), operator+=(), and Signal().

6.4.6 Member Data Documentation

6.4.6.1 index_t Signal::counter

general-purpose variable for external use.

Definition at line 104 of file Signal.h.

Referenced by play().

6.4.6.2 container_t Signal::data [private]

Holds the signal samples.

Definition at line 400 of file Signal.h.

Referenced by array(), delay(), filter(), gain(), l_inf_norm(), operator+=(), operator[](), samples(), set_samplerate(), set_size(), and Signal().

6.4.6.3 Signal::DFTDriver Signal::dft [static]

Single instance of the DFTDriver class.

Definition at line 397 of file Signal.h.

Referenced by filter().

6.4.6.4 int Signal::srate [private]

Signal sample rate in Hertz.

Definition at line 401 of file Signal.h.

Referenced by delay(), filter(), operator+=(), play(), samplerate(), set_samplerate(), and Signal().

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

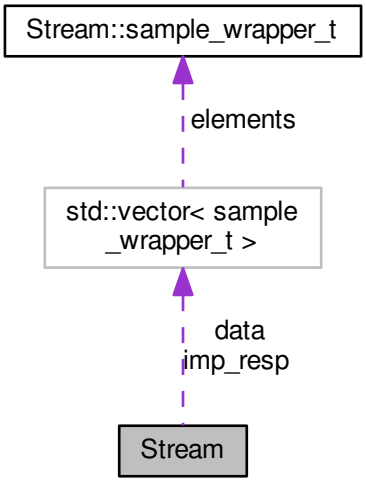
- Signal.h (v0.1.1-17-gc9cc97d)
- Signal.cpp (v0.1.1-17-gc9cc97d)

6.5 Stream Class Reference

Represents an input/output stream of audio samples.

`#include <Stream.h>`

Collaboration diagram for Stream:



Classes

- struct sample_wrapper_t

  A structured type for holding a single sample. Will be simplified later.

Public Types

- typedef float sample_t

  The type for holding each signal sample.
- typedef unsigned long index_t

  The type for holding each signal sample index.
- typedef std::vector
  < sample_wrapper_t > container_t

  The type for holding the whole vector of signal samples.

Public Member Functions

- sample_t read ()

  Returns the next audio sample.
- container_t::const_iterator get_last_n (index_t n)

  Returns an "array" with the last n samples.
- sample_t get_filtered_sample ()

  Returns a sample from the stream echoed by the impulse response.
- void write (sample_t s)

  Writes an audio sample to the stream.
- Stream ()

  Initializes important values.
- void echo (unsigned sleep=0)

  Runs the stream with predefined scenario parameters.
- void set_delay (unsigned msec)

  Sets the delay parameter, given in miliseconds.
- void set_filter (container_t h)

  Sets the room impulse response.
- void dump_state (const container_t speaker_buf) const

  Used for debugging, together with `simulate`
- void simulate ()

  Simulates a PortAudio session, used for debugging.

Static Public Attributes

- static const unsigned samplerate = 11025

  The stream's rate in samples per second.
- static const size_t buf_size = 8∗samplerate

  The number of data samples held internally be the stream structure.

Private Member Functions

- sample_t & operator[] (index_t index)

  Returns a sample.
- const sample_t & operator[] (index_t index) const

  Returns a "read-only" sample.

Private Attributes

- index_t delay_samples

  The delay of the communication channel, measured in samples.
- container_t data

  The container for holding the internal memory of the data structure.
- container_t::iterator write_ptr

  An iterator to the next location to be written on the stream.
- container_t::const_iterator read_ptr

  An iterator to the next location to be read from the stream.
- const container_t::const_iterator semantic_end

  An iterator to the middle of the vector `data`
- container_t imp_resp

6.5.1 Detailed Description

Represents an input/output stream of audio samples.

Holds data and provides routines for dealing with streams that represent communication systems with echo. Currently, all Streams are implemented as a circular array of single-precision floating-point samples. Streams are aware of their sample rates.

The stream uses the circular array as an internal memory. The array holds `2 * buf_size` audio samples. When we tell the stream to start running, the *write pointer* (`write_ptr`) points to the first element of this array. Everytime we receive a new audio sample from the microphone (through PortAudio), we write it to the location pointed to by the write pointer, and also to the location pointed to by `write_ptr + buf_size`. This way, we will always have a doubled vector of samples. When the write pointer reaches the middle of the circular vector, that is, the `buf_size+1`-th element, it rewinds back to the first element. A useful diagram is presented in the description for the `data` element.

Whenever we need a new audio sample to playback (which happens whenever we receive a new sample – the audio input and output are coerent), we read it from the the location pointed to by the *read pointer* (`read_ptr`), and increment the read pointer (rewinding it if necessary).

By calling the `set_delay()` method, we place the read pointer at a specified number of samples behind the write pointer, so that running the stream makes it echo everything it "hears".

- The `data` member of the class holds the circular array.
- `semantic_end` is a pointer to the "'buf_size+1'-th element". For looping through "all" the samples, we should do `for (iterator = data.begin(); iterator != semantic_end; ++iterator) {}`
- For writing samples in the manner specified above in second paragraph, we use the `write()` routine.
- Although there is an analogous `read()` routine, if we want the audio to pass through the room impulse response, we use the `get_filtered_sample()` method instead. It modifies the read pointer in the same way as `read()` does.

Definition at line 64 of file Stream.h.

6.5.2 Member Typedef Documentation

6.5.2.1 typedef std::vector<sample_wrapper_t> Stream::container_t

The type for holding the whole vector of signal samples.

Definition at line 83 of file Stream.h.

6.5.2.2 typedef unsigned long Stream::index_t

The type for holding each signal sample index.

Definition at line 72 of file Stream.h.

6.5.2.3 typedef float Stream::sample_t

The type for holding each signal sample.

Definition at line 69 of file Stream.h.

6.5.3 Constructor & Destructor Documentation

6.5.3.1 Stream::Stream ( ) [inline]

Initializes important values.

Constructs a scenario in which there is no delay, and the impulse response has one sample of value zero. Also acquires memory for the data structure.

See Also

data
read

Definition at line 203 of file Stream.h.

References buf_size, and samplerate.

6.5.4 Member Function Documentation

6.5.4.1 void Stream::dump_state (const container_t *speaker_buf*) const

Used for debugging, together with `simulate`

This function is called be `simulate()` to print to the screen the current state of the simulated environment. It prints the internal state of the Stream object and the samples that have been written to the output.

Parameters

in	*speaker_buf*	A vector with the samples that have been sent to the output (speaker).

See Also

simulate

Definition at line 176 of file Stream.cpp.

References data.

Referenced by simulate().

6.5.4.2 void Stream::echo (unsigned *sleep* = 0)

Runs the stream with predefined scenario parameters.

This is one of the main methods in the Stream class. It runs the stream, simulating a communications environment in which the user listens to echoes of his own voice.

Creates a PortAudio session for audio I/O. If *sleep* is not zero, we only run the stream for the time duration specified in miliseconds. If it's zero, we run the stream until something kills the process.

To use this method, you should first set the scenario parameters using the methods `set_filter` and `set_delay`.

Parameters

`in`	*sleep*	The duration, in miliseconds, in which to run the stream.

Exceptions

std::runtime_error	if any of the PortAudio steps fail (check the source code)

See Also

Stream
stream_callback

Definition at line 104 of file Stream.cpp.

References samplerate, set_delay(), and stream_callback().

Referenced by main().

6.5.4.3 sample_t Stream::get_filtered_sample ( ) [inline]

Returns a sample from the stream echoed by the impulse response.

Just like `read()`, but returns a sample from the signal that was convolved with the room impulse response.

Returns

a sample of the echoed signal

See Also

read
read_ptr

Definition at line 163 of file Stream.h.

References get_last_n(), and imp_resp.

Referenced by stream_callback().

6.5.4.4 container_t::const_iterator Stream::get_last_n (index_t *n*) [inline]

Returns an "array" with the last *n* samples.

Makes a pointer (actually, an iterator) to the n-th pointer behind `read_ptr`. Handles the case in which the range $[$`read_ptr` $-$ `n`, `read_ptr`$[$ crosses the end of the circular structure. That is, for any valid value of `read_ptr`, this function can be called with any `n` $\in [0,$`buf_size`$]$. A valid `read_ptr` is one such that `read_ptr - data.begin()` is in the range $[0,$`buf_size`$[$.

Parameters

`in`	*n*	The size, in samples, of the "array" that is returned

Returns

an iterator pointing to the last *n* samples.

See Also

data

Definition at line 143 of file Stream.h.

References buf_size, data, read_ptr, and semantic_end.

Referenced by get_filtered_sample().

6.5.4.5 sample_t& Stream::operator[] (index_t *index*) [inline], [private]

Returns a sample.

Gets a sample of the signal. Used only in debugging, when we want a snapshot of the stream internal data.

Parameters

in	*index*	The index of the desired sample.

Returns

a reference to the sample.

Definition at line 324 of file Stream.h.

References data.

6.5.4.6 const sample_t& Stream::operator[] (index_t *index*) const [inline], [private]

Returns a "read-only" sample.

Just like the "read-write" version, but returns a const reference to a sample.

Parameters

in	*index*	The index of the desired sample.

Returns

a const reference to the sample.

Definition at line 340 of file Stream.h.

References data.

6.5.4.7 sample_t Stream::read ( ) [inline]

Returns the next audio sample.

Reads the audio sample pointed to by `read_ptr` and makes the pointer indicate the next sample. Handles the case in which the pointer must be rewinded, because the data structure is circular.

Returns

the next audio sample in line

See Also

data
write

Definition at line 118 of file Stream.h.

References data, read_ptr, and semantic_end.

6.5.4.8 void Stream::set_delay (unsigned *msec*) [inline]

Sets the delay parameter, given in miliseconds.

Calculates how many samples are equivalent to the specified delay, and moves the read pointer to that many samples behind the write pointer.

This function doesn't check whether the given delay is valid, so that the application must be sure that the delay `msec` is such that $\texttt{samplerate} \cdot \texttt{msec} \leq 1000 \cdot \texttt{buf\_size}$

Parameters

in	*msec*	The time delay, specified in miliseconds

See Also

delay_samples
read_ptr

Definition at line 232 of file Stream.h.

References buf_size, data, delay_samples, read_ptr, samplerate, and write_ptr.

Referenced by echo(), main(), and simulate().

6.5.4.9 void Stream::set_filter (container_t *h*)

Sets the room impulse response.

This function just sets the internal copy of the room impulse response (RIR) samples to be equal to the one specified.

Parameters

in	*h*	A vector containing the RIR samples

Definition at line 161 of file Stream.cpp.

References imp_resp.

Referenced by main().

6.5.4.10 void Stream::simulate ( )

Simulates a PortAudio session, used for debugging.

This function simulates the PortAudio functioning by calling the callback function some four times, each time providing it with different in/out buffers.

See Also

dump_state

Definition at line 198 of file Stream.cpp.

References dump_state(), set_delay(), and stream_callback().

Referenced by main().

6.5.4.11 void Stream::write (sample_t *s*) [inline]

Writes an audio sample to the stream.

Writes a sample to the data structure so that it can be later read by `read()` or `get_last_n()`. Writes it twice (one in each copy of the data structure), so that the list is always "doubled" to make it look "circular".

Handles the case in which the pointer must be rewinded.

Parameters

in	*s*	The value of sample to be written to the stream's internal memory

See Also

data
read

Definition at line 186 of file Stream.h.

References buf_size, data, semantic_end, and write_ptr.

Referenced by stream_callback().

6.5.5 Member Data Documentation

6.5.5.1 const size_t Stream::buf_size = 8∗samplerate [static]

The number of data samples held internally be the stream structure.

The actual size of the vector used to hold the samples is `2*samplerate`, in order to make the data structure "look" circular.

Definition at line 94 of file Stream.h.

Referenced by get_last_n(), set_delay(), Stream(), and write().

6.5.5.2 container_t Stream::data [private]

The container for holding the internal memory of the data structure.

A C++ vector that holds the internal memory representation of the stream. The vector contains *two* copies of each audio sample. It can be though of as a concatenation of two identical vectors, each equal to a "circular buffer" of size `buf_size` that is used to keep track of the stream's state.

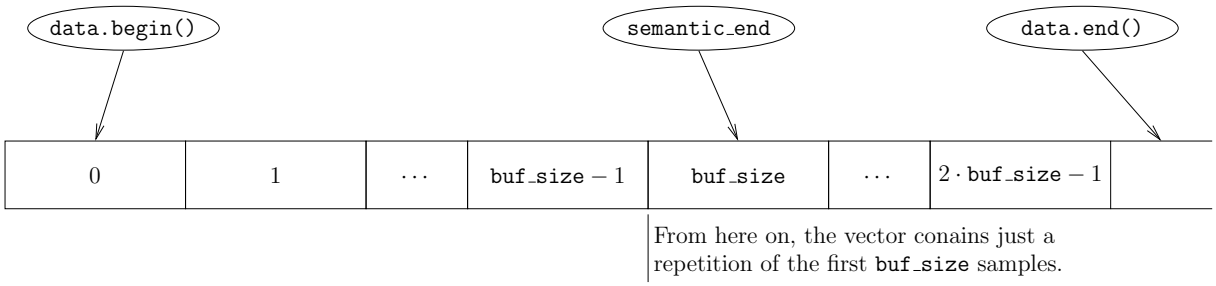


Figure 6.1: The 'data' vector structure

See Also

read
get_last_n
write
Stream

Definition at line 272 of file Stream.h.

Referenced by dump_state(), get_last_n(), operator[](), read(), set_delay(), and write().

6.5.5.3 index_t Stream::delay_samples [private]

The delay of the communication channel, measured in samples.

Definition at line 254 of file Stream.h.

Referenced by set_delay().

6.5.5.4 container_t Stream::imp_resp [private]

Definition at line 313 of file Stream.h.

Referenced by get_filtered_sample(), and set_filter().

6.5.5.5 container_t::const_iterator Stream::read_ptr [private]

An iterator to the next location to be read from the stream.

A sample should be read from the stream like:

```
sample = *read_ptr++;
if (read_ptr == semantic_end) read_ptr = data.begin();
```

See Also

read
semantic_end
write_ptr

Definition at line 303 of file Stream.h.

Referenced by get_last_n(), read(), and set_delay().

6.5.5.6 const unsigned Stream::samplerate = 11025 [static]

The stream's rate in samples per second.

Definition at line 87 of file Stream.h.

Referenced by echo(), set_delay(), and Stream().

6.5.5.7 const container_t::const_iterator Stream::semantic_end [private]

An iterator to the middle of the vector `data`

Points to the first repeating element in `data`. See the diagram provided in the `data` element description and the explanation in the `Stream` class description.

Definition at line 311 of file Stream.h.

Referenced by get_last_n(), read(), and write().

6.5.5.8 container_t::iterator Stream::write_ptr [private]

An iterator to the next location to be written on the stream.

A sample should be written to the stream like:

```
*write_ptr = sample;
*(write_ptr++ + buf_size) = sample;
if (write_ptr == semantic_end) write_ptr = data.begin();
```

It is important to write the sample to both locations (`write_ptr` and `write_ptr + buf_size`) in case we read from the stream through functions like `get_last_n()`.

See Also

write
semantic_end
read_ptr

Definition at line 290 of file Stream.h.

Referenced by set_delay(), and write().

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

- Stream.h (v0.1.1-17-gc9cc97d)
- Stream.cpp (v0.1.1-17-gc9cc97d)

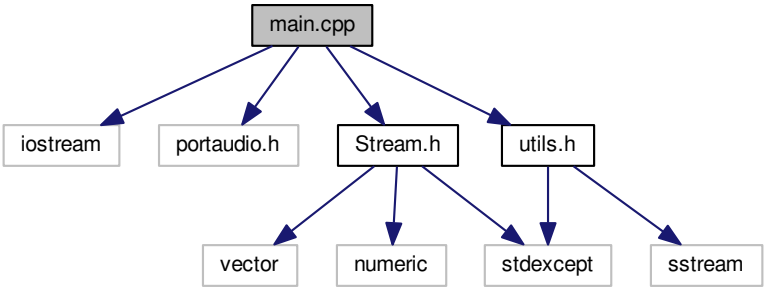
Chapter 7

File Documentation

7.1 main.cpp File Reference

```
#include <iostream>
#include <portaudio.h>
#include "Stream.h"
#include "utils.h"
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[])

  `main()` function.

7.1.1 Detailed Description

Holds the `main()` function.

Author

Pedro Angelo Medeiros Fonini

Definition in file main.cpp.

7.1.2 Function Documentation

7.1.2.1 int main (int *argc*, char * *argv[]*)

`main()` function.

No command-line parameters.

This function:

1. Prints version info
2. Creates an i/o stream to represent the communication channel with echo
3. Creates a room impulse response
4. Assigns the created impulse response to the stream
5. Assigns a value of 300ms to the stream's delay echo
6. Runs the stream

Parameters

in	*argc*	argument count (unused)
in	*argv*	argument values (unused)

Returns

0 if no errors

Definition at line 46 of file main.cpp.

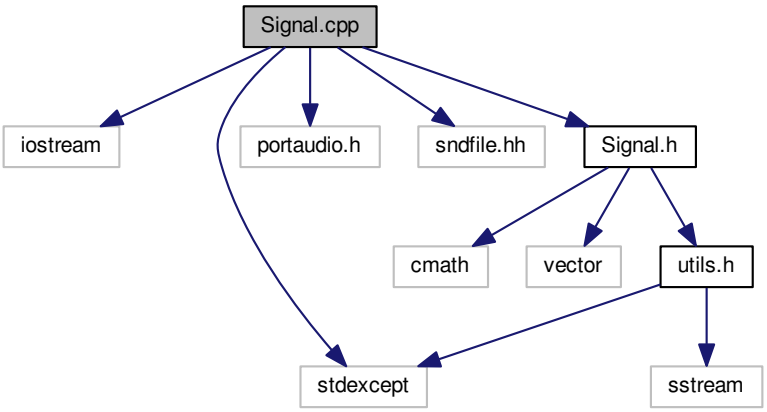
References Stream::echo(), portaudio_end(), portaudio_init(), Stream::set_delay(), Stream::set_filter(), and Stream::simulate().

7.2 README.md File Reference

7.3 Signal.cpp File Reference

```
#include <iostream>
#include <stdexcept>
#include <portaudio.h>
#include <sndfile.hh>
#include "Signal.h"
```

Include dependency graph for Signal.cpp:



Functions

- static int signal_callback (const void *in_buf, void *out_buf, unsigned long frames_per_buf, const PaStreamCallbackTimeInfo *time_info, PaStreamCallbackFlags status_flags, void *user_data)

  PortAudio callback function.

7.3.1 Detailed Description

Holds the implementation of the `Signal` class.

Author

Pedro Angelo Medeiros Fonini

Definition in file Signal.cpp.

7.3.2 Function Documentation

7.3.2.1 static int signal_callback (const void * *in_buf*, void * *out_buf*, unsigned long *frames_per_buf*, const PaStreamCallbackTimeInfo * *time_info*, PaStreamCallbackFlags *status_flags*, void * *user_data*) [static]

PortAudio callback function.

The PortAudio library implements the stream playback using *callback* functions. These functions get called at interrupt time whenever PortAudio needs a new buffer of samples to pass to the hardware. Callback functions should not take long to return; in particular, they should **not** throw or catch exceptions, or do I/O.

This callback function just reads a given `Signal` and passes its samples to PortAudio.

Parameters

in	*in_buf*	Pointer to a buffer of samples retrieved from an input audio device. This parameter is unused because we're not reading from any device.
out	*out_buf*	Pointer to a buffer where the callback function will store samples to be given to an output audio device.
in	*frames_per_buf*	Number of samples we will store in the buffer. This is actually the number of *frames*, which in turn is equal to number of samples because we're working with mono-channel signals.
in	*time_info*	PortAudio time information. (unused)
in	*status_flags*	PortAudio status flags. (unused)
in,out	*user_data*	Pointer to an arbitrary data-holder passed to the stream open function. We use this to get the signal samples, and to keep track of where in the signal we are (using the `Signal::counter` auxiliary member).

See Also

Signal::play

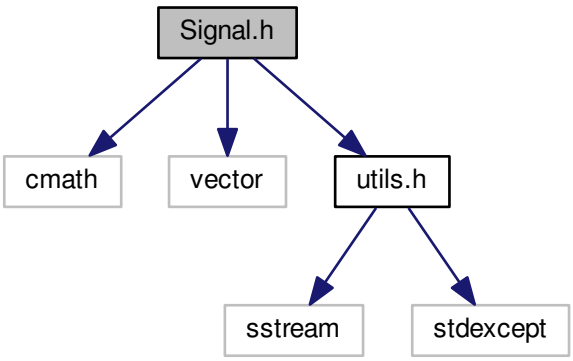
Definition at line 275 of file Signal.cpp.

Referenced by Signal::play().

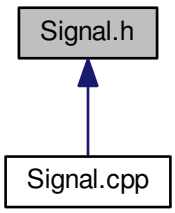
7.4 Signal.h File Reference

```
#include <cmath>
#include <vector>
#include "utils.h"
```

Include dependency graph for Signal.h:



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



Classes

- class Signal

  A time- or frequency-domain signal.
- class Signal::DFTDriver

  A class for providing discrete Fourier transform capabilities.

Functions

- Signal operator+ (Signal lhs, const Signal &rhs)

  Adds two signals.

7.4.1 Detailed Description

Holds the interface to the `Signal` class.

Author

Pedro Angelo Medeiros Fonini

Definition in file Signal.h.

7.4.2 Function Documentation

7.4.2.1 Signal operator+ (Signal *lhs,* const Signal & *rhs*) [inline]

Adds two signals.

See Also

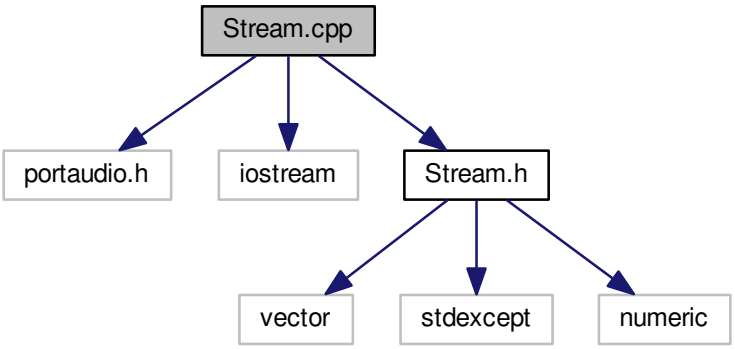
Signal::operator+=

Definition at line 409 of file Signal.h.

7.5 Stream.cpp File Reference

```
#include <portaudio.h>
#include <iostream>
#include "Stream.h"
```

Include dependency graph for Stream.cpp:



Functions

- static int stream_callback (const void *in_buf, void *out_buf, unsigned long frames_per_buf, const PaStreamCallbackTimeInfo *time_info, PaStreamCallbackFlags status_flags, void *user_data)

  Callback function for dealing with PortAudio.

7.5.1 Detailed Description

Holds the implementation of the `Stream` class.

Author

Pedro Angelo Medeiros Fonini

Definition in file Stream.cpp.

7.5.2 Function Documentation

7.5.2.1 static int stream_callback (const void * *in_buf*, void * *out_buf*, unsigned long *frames_per_buf*, const PaStreamCallbackTimeInfo * *time_info*, PaStreamCallbackFlags *status_flags*, void * *user_data*) [static]

Callback function for dealing with PortAudio.

See the description for the `signal_callback()` function, in the Signal.cpp file for information on how this function accomplishes audio I/O, together with PortAudio.

Parameters

in	*in_buf*	Pointer to a buffer of samples retrieved from an input audio device. We read these samples into the stream, at the location pointed to by the Stream's Stream::write_ptr.
out	*out_buf*	Pointer to a buffer where the callback function will store samples to be given to an output audio device. We write to this location the samples we get from the stream through the Stream::read_ptr pointer.
in	*frames_per_buf*	Number of samples we will write/read to/from the PortAudio buffers. This is actually the number of *frames*, which in turn is equal to number of samples because we're working with mono-channel signals.
in	*time_info*	PortAudio time information. (unused)
in	*status_flags*	PortAudio status flags. (unused)
in,out	*user_data*	Pointer to an arbitrary data-holder passed to the stream open function. In this case, this is a pointer to the Stream object.

See Also

Stream::echo

Definition at line 61 of file Stream.cpp.

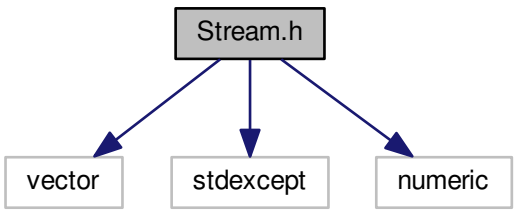
References Stream::get_filtered_sample(), and Stream::write().

Referenced by Stream::echo(), and Stream::simulate().

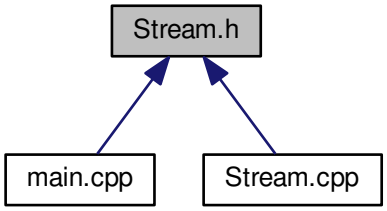
7.6 Stream.h File Reference

```
#include <vector>
#include <stdexcept>
#include <numeric>
```

Include dependency graph for Stream.h:



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



Classes

- class Stream

  Represents an input/output stream of audio samples.

- struct Stream::sample_wrapper_t

  A structured type for holding a single sample. Will be simplified later.

7.6.1 Detailed Description

Holds the interface to the `Stream` class.

Author

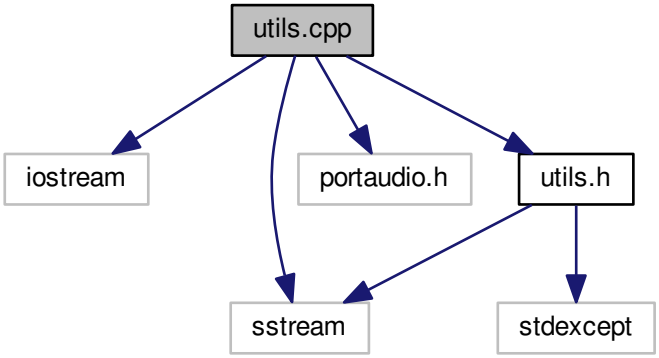
Pedro Angelo Medeiros Fonini

Definition in file Stream.h.

7.7 utils.cpp File Reference

```
#include <iostream>
#include <sstream>
#include <portaudio.h>
#include "utils.h"
```

Include dependency graph for utils.cpp:



Functions

- void portaudio_init (bool list_devices)
  Initialize PortAudio.
- void portaudio_end ()
  Close PortAudio.

7.7.1 Detailed Description

Holds convenient routines.

Author

Pedro Angelo Medeiros Fonini

Definition in file utils.cpp.

7.7.2 Function Documentation

7.7.2.1 void portaudio_end ( )

Close PortAudio.

Ends a PortAudio session.

Exceptions

std::runtime_error	if PortAudio closing fails.

See Also

`portaudio_init()`

Definition at line 84 of file utils.cpp.

Referenced by main().

7.7.2.2 void portaudio_init (bool *list_devices*)

Initialize PortAudio.

Initializes a PortAudio session. Also prints out a list of available devices that PortAudio sees., if requested.

Parameters

in	*list_devices*	Whether or not to print the device list.

Exceptions

std::runtime_error	if PortAudio initialization fails.

See Also

`portaudio_end()`

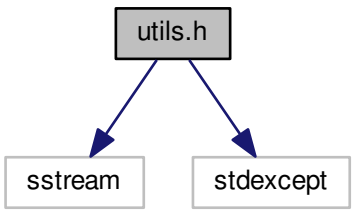
Definition at line 42 of file utils.cpp.

Referenced by main().

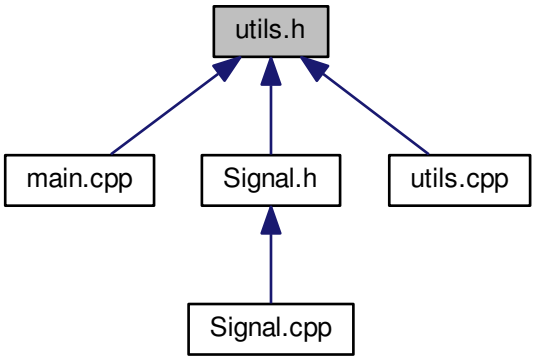
7.8 utils.h File Reference

```
#include <sstream>
#include <stdexcept>
```

Include dependency graph for utils.h:



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



Classes

- class FileError
  A runtime exception while trying to process a file.

Functions

- void portaudio_init (bool list_devices=false)
  Initialize PortAudio.
- void portaudio_end ()
  Close PortAudio.

Variables

- static const double TAU = 6.283185307179586477
  Shorthand for the number 2π.

7.8.1 Detailed Description

Holds convenient definitions and other utilities.

Author

Pedro Angelo Medeiros Fonini

Definition in file utils.h.

7.8.2 Function Documentation

7.8.2.1 void portaudio_end ( )

Close PortAudio.

Ends a PortAudio session.

Exceptions

std::runtime_error	if PortAudio closing fails.

See Also

`portaudio_init()`

Definition at line 84 of file utils.cpp.

Referenced by main().

7.8.2.2 void portaudio_init (bool *list_devices*)

Initialize PortAudio.

Initializes a PortAudio session. Also prints out a list of available devices that PortAudio sees., if requested.

Parameters

in	*list_devices*	Whether or not to print the device list.

Exceptions

std::runtime_error	if PortAudio initialization fails.

See Also

`portaudio_end()`

Definition at line 42 of file utils.cpp.

Referenced by main().

7.8.3 Variable Documentation

7.8.3.1 const double TAU = 6.283185307179586477 [static]

Shorthand for the number 2π.

Useful in the generation of the table of sines and cosines for the Signal::DFTDriver class, for example.

Definition at line 54 of file utils.h.

Referenced by Signal::DFTDriver::initialize_costbl(), and Signal::DFTDriver::initialize_sintbl().

Index