

Andrew and Nick's Project

Generated by Doxygen 1.8.8

Fri Apr 15 2016 02:50:06

Contents

1	Namespace Index	1
1.1	Namespace List	1
2	Class Index	3
2.1	Class List	3
3	File Index	5
3.1	File List	5
4	Namespace Documentation	7
4.1	vaso Namespace Reference	7
4.1.1	Detailed Description	7
4.1.2	Enumeration Type Documentation	8
4.1.2.1	Side	8
4.1.3	Function Documentation	8
4.1.3.1	absolute	8
4.1.3.2	average	8
4.1.3.3	average	9
4.1.3.4	average	9
4.1.3.5	CurrentDataName	9
4.1.3.6	decibels	10
4.1.3.7	diff	10
4.1.3.8	fft	10
4.1.3.9	InitialDataName	11
4.1.3.10	mag	11
4.1.3.11	max	12
4.1.3.12	PatientName	12
4.1.3.13	play	12
4.1.3.14	Process	12
4.1.3.15	ReadParams	14
4.1.3.16	smooth	14
4.1.3.17	WriteParams	14

4.1.4	Variable Documentation	15
4.1.4.1	PATIENT_PATH	15
5	Class Documentation	17
5.1	DataParams Struct Reference	17
5.1.1	Detailed Description	17
5.1.2	Member Data Documentation	17
5.1.2.1	freq	17
5.1.2.2	noise	17
5.2	Maximum Struct Reference	17
5.2.1	Detailed Description	18
5.2.2	Member Data Documentation	18
5.2.2.1	index	18
5.2.2.2	value	18
5.3	ThreadParams Struct Reference	18
5.3.1	Detailed Description	18
5.3.2	Member Data Documentation	18
5.3.2.1	counter	18
5.3.2.2	data	18
5.3.2.3	recCount	18
5.3.2.4	results	19
5.3.2.5	sampleCount	19
5.3.2.6	sampleFreq	19
6	File Documentation	21
6.1	src/definitions.hpp File Reference	21
6.1.1	Macro Definition Documentation	23
6.1.1.1	ENUM	23
6.1.1.2	ERROR	23
6.1.1.3	REC_COUNT	23
6.1.1.4	SAMPLE_COUNT	23
6.1.1.5	SAMPLE_FREQ	23
6.1.2	Typedef Documentation	23
6.1.2.1	byte	23
6.1.2.2	cfloat32	23
6.1.2.3	float32	24
6.1.2.4	float64	24
6.1.2.5	sint16	24
6.1.2.6	sint32	24
6.1.2.7	sint64	24
6.1.2.8	sint8	24

6.1.2.9	uint16	24
6.1.2.10	uint32	24
6.1.2.11	uint64	24
6.1.2.12	uint8	24
6.2	definitions.hpp	24
6.3	src/fileio.hpp File Reference	25
6.4	fileio.hpp	26
6.5	src/main.cpp File Reference	27
6.5.1	Function Documentation	27
6.5.1.1	main	27
6.6	main.cpp	28
6.7	src/process.hpp File Reference	29
6.8	process.hpp	30
6.9	src/sigmath.hpp File Reference	31
6.10	sigmath.hpp	32
6.11	src/sound.hpp File Reference	34
6.12	sound.hpp	35
Index		36

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

vaso	Functions related to the file I/O use in this program	7
----------------------	---	-------------------

Chapter 2

Class Index

2.1 Class List

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

DataParams	17
Maximum	17
ThreadParams	18

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

src/ definitions.hpp	21
src/ fileio.hpp	25
src/ main.cpp	27
src/ process.hpp	29
src/ sigmath.hpp	31
src/ sound.hpp	34

Chapter 4

Namespace Documentation

4.1 vaso Namespace Reference

contains functions related to the file I/O use in this program

Enumerations

- enum [Side](#) { [Side::Left](#), [Side::Right](#) }

Functions

- std::string [CurrentDataName](#) ()
- std::string [InitialDataName](#) (auto dir)
- std::string [PatientName](#) ()
- [DataParams ReadParams](#) (auto filename)
- std::string [WriteParams](#) ([DataParams](#) params, auto filename)
- std::map< [Side](#), [DataParams](#) > [Process](#) (float32 data[[REC_COUNT](#)][[SAMPLE_COUNT](#)])
- void [absolute](#) (float32 *data, uint32 size)
- float32 [average](#) (float32 *data, uint32 size)
- [DataParams average](#) ([DataParams](#) *params, uint8 size)
- void [average](#) (float32 *data, float32 *avg, uint8 count, uint32 size)
- void [decibels](#) (float32 *data, uint32 size)
- void [diff](#) (float32 *data, uint32 size)
- void [fft](#) (cfloat32 *data, uint32 size)
- void [mag](#) (cfloat32 *orig, float32 *newmags, uint32 size)
- [Maximum max](#) (float32 *data, uint32 size)
- void [smooth](#) (float32 *data, uint32 size, uint16 order)
- void [play](#) (auto filename)

Variables

- const std::string [PATIENT_PATH](#) = "/home/pi/patients/"

4.1.1 Detailed Description

contains functions related to the file I/O use in this program

contains the function(s) relating to sound

contains the functions necessary to perform the mathematical operations required by this program

contains function(s) related to the program's threaded processing of audio data

This namespace contains all code related to this project.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Samuel Andrew Wisner, awisner94@gmail.com

Nicholas K. Nolan

4.1.2 Enumeration Type Documentation

4.1.2.1 `enum vaso::Side` [strong]

The side of the head to which a recording pertains.

Enumerator

Left

Right

Definition at line 65 of file [definitions.hpp](#).

4.1.3 Function Documentation

4.1.3.1 `void vaso::absolute (float32 * data, uint32 size)`

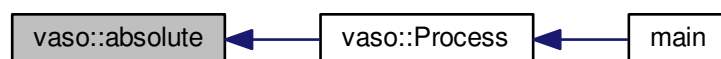
Ensures all elements in an array are positive. Note that this function replaces array elements if necessary. It does not populate a new array.

Parameters

<i>data</i>	the array whose elements must all be positive
<i>size</i>	the number of elements in the data array

Definition at line 141 of file [sigmath.hpp](#).

Here is the caller graph for this function:



4.1.3.2 `float32 vaso::average (float32 * data, uint32 size)`

Takes the average of all elements in an array

Parameters

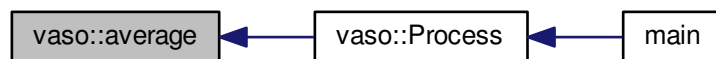
<i>data</i>	the array from which to compute the average
<i>size</i>	the number of elements in the data array

Returns

the computed average

Definition at line 145 of file [sigmath.hpp](#).

Here is the caller graph for this function:

4.1.3.3 **DataParams** vaso::average (**DataParams** * *params*, uint8 *size*)

Finds the averages of the elements of an array of [DataParams](#).

Parameters

<i>params</i>	the DataParams array
<i>size</i>	the number of elements in the DataParams array

Returns

a [DataParams](#) structure containing the average values of the structure's elements in the params array

Definition at line 149 of file [sigmath.hpp](#).

4.1.3.4 void vaso::average (float32 * *data*, float32 * *avg*, uint8 *count*, uint32 *size*)

Element-wise averaging along the first dimension of a two-dimensional array.

Parameters

<i>data</i>	the two-dimensional array containing [count] number of arrays in the first dimension and [size] number of each elements in the second dimension
<i>avg</i>	the array of size [size] containing the averaged values of each element
<i>count</i>	the number of arrays in the first dimension of data and will likely be a constant value of 3 in this program
<i>size</i>	the number of elements in the second dimension of data

Definition at line 153 of file [sigmath.hpp](#).

4.1.3.5 std::string vaso::CurrentDataName ()

Gets a data-based name to which the file(s) created in a session to be saved.

Returns

a partial (?) filename for the current session

Definition at line 26 of file [fileio.hpp](#).

4.1.3.6 void vaso::decibels (float32 * data, uint32 size)

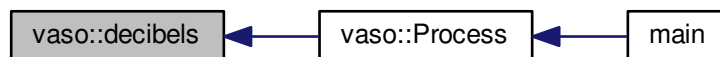
Converts an array of floats to "power decibels", i.e., $x[n] = 20 \cdot \log_{10}(x[n])$. The decibel values are written to the same array that contained the values to be converted. In other words, this function should perform an in-place, element-wise conversion.

Parameters

<i>data</i>	the array of values to be converted as well as the location where the converted values will be written
<i>size</i>	the number of elements in the data array

Definition at line 157 of file [sigmath.hpp](#).

Here is the caller graph for this function:

**4.1.3.7 void vaso::diff (float32 * data, uint32 size)**

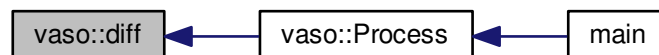
Computes the left-handed first derivative of a discrete signal. The first element will be 0.

Parameters

<i>data</i>	an array containing the discrete signal data
<i>size</i>	the number of elements in data

Definition at line 163 of file [sigmath.hpp](#).

Here is the caller graph for this function:

**4.1.3.8 void vaso::fft (cfloat32 * data, uint32 size)**

Replaces the values of an array of cfloat32's with the array's DFT using a decimation-in-frequency algorithm.

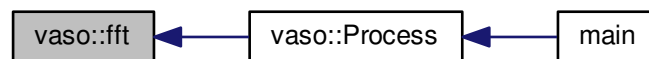
This code is based on code from http://rosettacode.org/wiki/Fast_Fourier_transform#C.↵2B.2B.

Parameters

<i>data</i>	the array whose values should be replaced with its DFT
<i>size</i>	the number of elements in the data array

Definition at line 167 of file [sigmath.hpp](#).

Here is the caller graph for this function:



4.1.3.9 `std::string vaso::InitialDataName (auto dir)`

Finds the filename of the oldest (i.e., baseline) data is saved.

Parameters

<i>dir</i>	the directory which contains all patient data
------------	---

Returns

the base (?) filename to which all baseline data was saved

Definition at line 37 of file [fileio.hpp](#).

4.1.3.10 `void vaso::mag (cfloat32 * orig, float32 * newmags, uint32 size)`

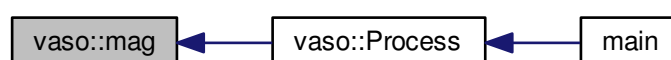
Computes the magitude of an array of complex numbers.

Parameters

<i>orig</i>	the array of complex numbers
<i>newmags</i>	an array to which the magitudes are to be written
<i>size</i>	the number of elements in orig and newmags

Definition at line 215 of file [sigmath.hpp](#).

Here is the caller graph for this function:



4.1.3.11 Maximum `vaso::max (float32 * data, uint32 size)`

Finds the maximum value in an array.

Parameters

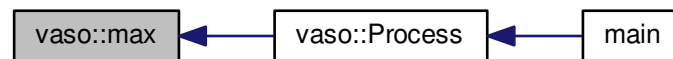
<i>data</i>	the array whose maximum value is to be found
<i>uint32</i>	size the number of elements in the data array

Returns

the maximum value and its index in a [Maximum](#) structure

Definition at line 219 of file [sigmath.hpp](#).

Here is the caller graph for this function:



4.1.3.12 `std::string vaso::PatientName ()`

Prompts a user to enter a first, middle, and last name for a patients and creates a directory (if necessary) in which all of a patient's data can be saved.

Must warn a user if the patient folder does not already exist in order to prevent missaving data.

Returns

the directory under which all patient data is saved

Definition at line 51 of file [fileio.hpp](#).

4.1.3.13 `void vaso::play (auto filename)`

Plays a WAVE file in a loop in a non-blocking manner.

Parameters

<i>filename</i>	the absolute or relative path to the WAVE file
-----------------	--

Definition at line 19 of file [sound.hpp](#).

4.1.3.14 `std::map<Side, DataParams> vaso::Process (float32 data[REC_COUNT][SAMPLE_COUNT])`

Processes the recorded audio. Meant to be run in a separate thread as the recordings are being made. This function assumes that the left-side recordings will be made first.

It should be noted that is algorithm is considered the intellectual property of Andrew Wisner and Nicholas Nolan. The "algorithm" is defined as the use of 1) the frequency drop-off and/or 2) a noise value from the frequency band above the drop-off frequency in order to diagnose (with or without other factors and parameters) the presence of

a vasospasm in a patient. By faculty members and/or students in the UAB ECE department using this algorithm, they agree that the presentation of their code or project that uses this algorithm by anyone directly or indirectly related to the code or project, whether verbally or in writing, will reference the development of the initial algorithm by Andrew Wisner and Nicholas Nolan. Furthermore, a failure to meet this stipulation will warrant appropriate action by Andrew Wisner and/or Nicholas Nolan. It should be understood that the purpose of this stipulation is not to protect proprietary rights; rather, it is to help ensure that the intellectual property of the aforementioned is protected and is neither misrepresented nor claimed implicitly or explicitly by another individual.

data two-dimensional array (first dimension whole recordings, second dimension samples in a recording) that will contain all recorded audio

REC_COUNT the number of recordings (left and right together) to be made

Parameters

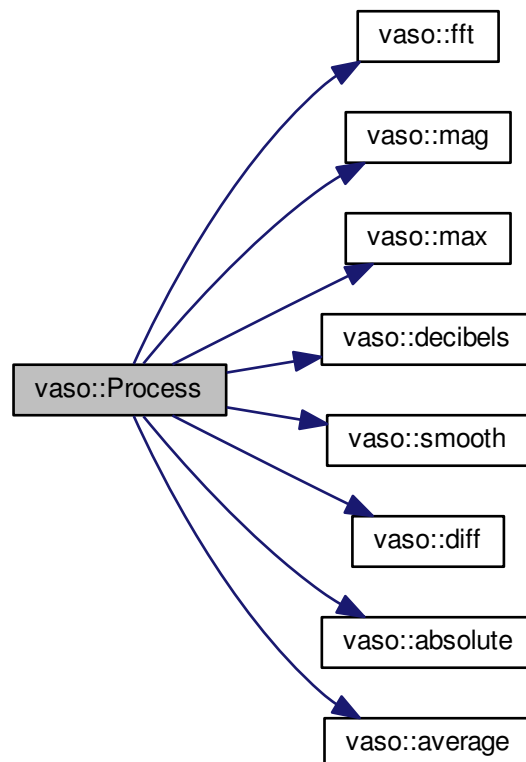
<i>SAMPLE_COUNT</i> <i>UNT</i>	the number of samples in each recording. MUST be a power of two.
<i>SAMPLE_FREQ</i>	the sampling frequency in Hz or Samples/second

Returns

a map of the averaged left- and right-side parameters in [DataParams](#) structures

Definition at line 54 of file [process.hpp](#).

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.15 DataParams vaso::ReadParams (auto filename)

Reads the previously computed parameters found in the specified file.

Parameters

<i>filename</i>	the absolute or relative path to the file containing the patient data to read
-----------------	---

Returns

the patient parameters read

Definition at line 64 of file [fileio.hpp](#).

4.1.3.16 void vaso::smooth (float32 * data, uint32 size, uint16 order)

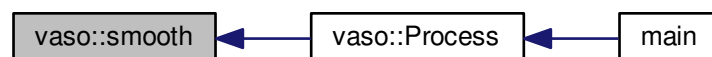
Applies an nth-order moving-average filter to a discrete signal.

Parameters

<i>data</i>	the array containing the signal to which the filter should be applied
<i>size</i>	the number of elements in the data array
<i>order</i>	the order of the filter

Definition at line 223 of file [sigmath.hpp](#).

Here is the caller graph for this function:



4.1.3.17 std::string vaso::WriteParams (DataParams params, auto filename)

Writes the parameters to the specified file.

Parameters

<i>params</i>	
---------------	--

Definition at line 73 of file [fileio.hpp](#).

4.1.4 Variable Documentation

4.1.4.1 `const std::string vaso::PATIENT_PATH = "/home/pi/patients/"`

Absolute path to the folder containing the patients' data

Definition at line 18 of file [fileio.hpp](#).

Chapter 5

Class Documentation

5.1 DataParams Struct Reference

```
#include <definitions.hpp>
```

Public Attributes

- [float32 freq](#)
- [float32 noise](#)

5.1.1 Detailed Description

Contains the calculated results from processing the audio recordings.

Definition at line [44](#) of file [definitions.hpp](#).

5.1.2 Member Data Documentation

5.1.2.1 float32 DataParams::freq

Definition at line [45](#) of file [definitions.hpp](#).

5.1.2.2 float32 DataParams::noise

Definition at line [46](#) of file [definitions.hpp](#).

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

- [src/definitions.hpp](#)

5.2 Maximum Struct Reference

```
#include <definitions.hpp>
```

Public Attributes

- [float32 value](#)
- [uint32 index](#)

5.2.1 Detailed Description

Contains the maximum value found in an array and the value's index in that array.

Definition at line 53 of file [definitions.hpp](#).

5.2.2 Member Data Documentation

5.2.2.1 uint32 Maximum::index

Definition at line 55 of file [definitions.hpp](#).

5.2.2.2 float32 Maximum::value

Definition at line 54 of file [definitions.hpp](#).

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

- [src/definitions.hpp](#)

5.3 ThreadParams Struct Reference

```
#include <definitions.hpp>
```

Public Attributes

- [float32** data](#)
- [uint8 recCount](#)
- [uint32 sampleCount](#)
- [uint32 sampleFreq](#)
- [uint8 * counter](#)
- [std::map< vaso::Side, DataParams > results](#)

5.3.1 Detailed Description

Contains the information needed by the thread that executes the [Process\(\)](#) function.

Definition at line 72 of file [definitions.hpp](#).

5.3.2 Member Data Documentation

5.3.2.1 uint8* ThreadParams::counter

Definition at line 77 of file [definitions.hpp](#).

5.3.2.2 float32** ThreadParams::data

Definition at line 73 of file [definitions.hpp](#).

5.3.2.3 uint8 ThreadParams::recCount

Definition at line 74 of file [definitions.hpp](#).

5.3.2.4 `std::map<vaso::Side, DataParams> ThreadParams::results`

Definition at line 78 of file [definitions.hpp](#).

5.3.2.5 `uint32 ThreadParams::sampleCount`

Definition at line 75 of file [definitions.hpp](#).

5.3.2.6 `uint32 ThreadParams::sampleFreq`

Definition at line 76 of file [definitions.hpp](#).

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

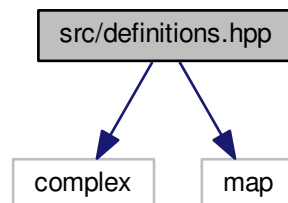
- [src/definitions.hpp](#)

Chapter 6

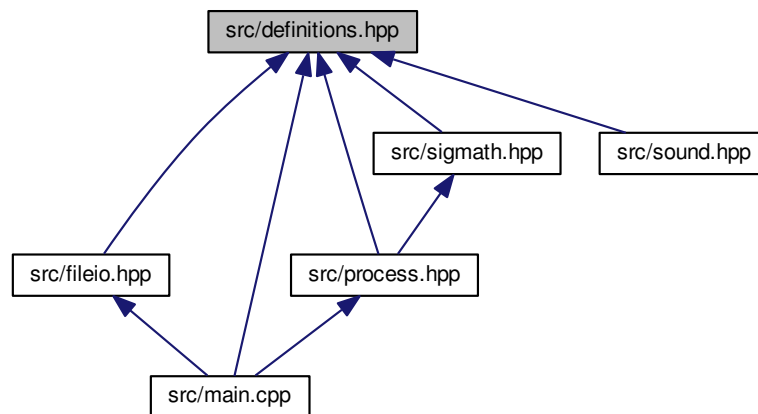
File Documentation

6.1 src/definitions.hpp File Reference

```
#include <complex>
#include <map>
Include dependency graph for definitions.hpp:
```



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



Classes

- struct [DataParams](#)
- struct [Maximum](#)
- struct [ThreadParams](#)

Namespaces

- [vaso](#)
contains functions related to the file I/O use in this program

Macros

- `#define` [ERROR](#) -1
Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations.
- `#define` [REC_COUNT](#) 8
- `#define` [SAMPLE_COUNT](#) 262144
- `#define` [SAMPLE_FREQ](#) 48000
- `#define` [ENUM](#) signed char

Typedefs

- typedef unsigned char [byte](#)
- typedef unsigned char [uint8](#)
- typedef signed char [sint8](#)
- typedef unsigned short [uint16](#)
- typedef signed short [sint16](#)
- typedef unsigned int [uint32](#)
- typedef signed int [sint32](#)
- typedef unsigned long long [uint64](#)

- typedef signed long long [sint64](#)
- typedef float [float32](#)
- typedef double [float64](#)
- typedef std::complex< [float32](#) > [cfloat32](#)

Enumerations

- enum [vaso::Side](#) { [vaso::Side::Left](#), [vaso::Side::Right](#) }

6.1.1 Macro Definition Documentation

6.1.1.1 #define ENUM signed char

Definition at line [18](#) of file [definitions.hpp](#).

6.1.1.2 #define ERROR -1

Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations.

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition at line [14](#) of file [definitions.hpp](#).

6.1.1.3 #define REC_COUNT 8

Definition at line [15](#) of file [definitions.hpp](#).

6.1.1.4 #define SAMPLE_COUNT 262144

Definition at line [16](#) of file [definitions.hpp](#).

6.1.1.5 #define SAMPLE_FREQ 48000

Definition at line [17](#) of file [definitions.hpp](#).

6.1.2 Typedef Documentation

6.1.2.1 typedef unsigned char byte

Definition at line [20](#) of file [definitions.hpp](#).

6.1.2.2 typedef std::complex<float32> cfloat32

Defines a type for complex float32's.

Definition at line [39](#) of file [definitions.hpp](#).

6.1.2.3 typedef float float32

Definition at line 33 of file [definitions.hpp](#).

6.1.2.4 typedef double float64

Definition at line 34 of file [definitions.hpp](#).

6.1.2.5 typedef signed short sint16

Definition at line 25 of file [definitions.hpp](#).

6.1.2.6 typedef signed int sint32

Definition at line 28 of file [definitions.hpp](#).

6.1.2.7 typedef signed long long sint64

Definition at line 31 of file [definitions.hpp](#).

6.1.2.8 typedef signed char sint8

Definition at line 22 of file [definitions.hpp](#).

6.1.2.9 typedef unsigned short uint16

Definition at line 24 of file [definitions.hpp](#).

6.1.2.10 typedef unsigned int uint32

Definition at line 27 of file [definitions.hpp](#).

6.1.2.11 typedef unsigned long long uint64

Definition at line 30 of file [definitions.hpp](#).

6.1.2.12 typedef unsigned char uint8

Definition at line 21 of file [definitions.hpp](#).

6.2 definitions.hpp

```
00001
00008 #ifndef definitions_H
00009 #define definitions_H
00010
00011 #include <complex>
00012 #include <map>
00013
00014 #define ERROR -1
00015 #define REC_COUNT 8
00016 #define SAMPLE_COUNT 262144
00017 #define SAMPLE_FREQ 48000
```

```

00018 #define ENUM signed char
00019
00020 typedef unsigned char byte;
00021 typedef unsigned char uint8;
00022 typedef signed char sint8;
00023
00024 typedef unsigned short uint16;
00025 typedef signed short sint16;
00026
00027 typedef unsigned int uint32;
00028 typedef signed int sint32;
00029
00030 typedef unsigned long long uint64;
00031 typedef signed long long sint64;
00032
00033 typedef float float32;
00034 typedef double float64;
00035
00039 typedef std::complex<float32> cfloat32;
00040
00044 typedef struct {
00045     float32 freq;
00046     float32 noise;
00047 } DataParams;
00048
00053 typedef struct {
00054     float32 value;
00055     uint32 index;
00056 } Maximum;
00057
00061 namespace vaso {
00065     enum class Side { Left, Right };
00066 }
00067
00072 typedef struct {
00073     float32** data;
00074     uint8 recCount;
00075     uint32 sampleCount;
00076     uint32 sampleFreq;
00077     uint8* counter;
00078     std::map<vaso::Side, DataParams> results;
00079 } ThreadParams;
00080
00081 #endif

```

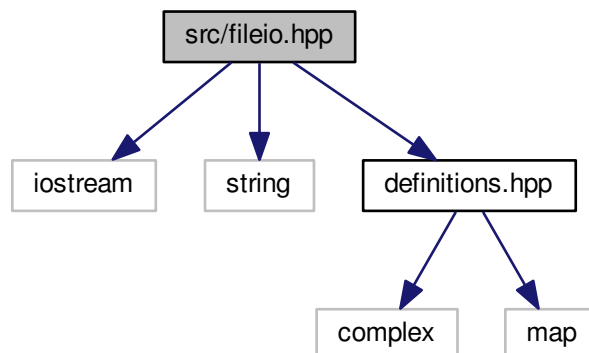
6.3 src/fileio.hpp File Reference

```

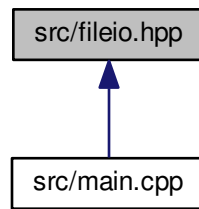
#include <iostream>
#include <string>
#include "definitions.hpp"

```

Include dependency graph for fileio.hpp:



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



Namespaces

- `vaso`
contains functions related to the file I/O use in this program

Functions

- `std::string vaso::CurrentDataName ()`
- `std::string vaso::InitialDataName (auto dir)`
- `std::string vaso::PatientName ()`
- `DataParams vaso::ReadParams (auto filename)`
- `std::string vaso::WriteParams (DataParams params, auto filename)`

Variables

- `const std::string vaso::PATIENT_PATH = "/home/pi/patients/"`

6.4 fileio.hpp

```

00001
00006 #ifndef fileio_H
00007 #define fileio_H
00008
00009 #include <iostream>
00010 #include <string>
00011
00012 #include "definitions.hpp"
00013
00014 namespace vaso {
00018     const std::string PATIENT_PATH = "/home/pi/patients/";
00019
00026     std::string CurrentDataName () {
00027     }
00028
00029     std::string InitialDataName(auto dir) {
00038     }
00039
00040     std::string PatientName () {
00052     }
00053
00054     DataParams ReadParams(auto filename) {
00065     }
00066
  
```



```

00067
00073     std::string WriteParams(DataParams params, auto filename) {
00074
00075     }
00076 }
00077
00078 #endif

```

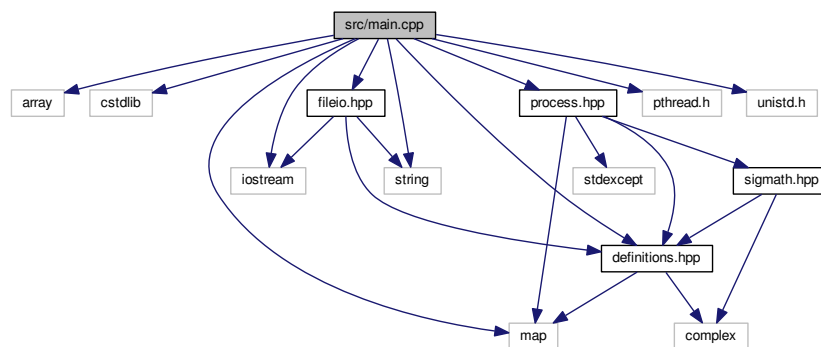
6.5 src/main.cpp File Reference

```

#include <array>
#include <cstdlib>
#include <iostream>
#include <map>
#include <pthread.h>
#include <string>
#include <unistd.h>
#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"

```

Include dependency graph for main.cpp:



Functions

- `int main (int argc, char **argv)`

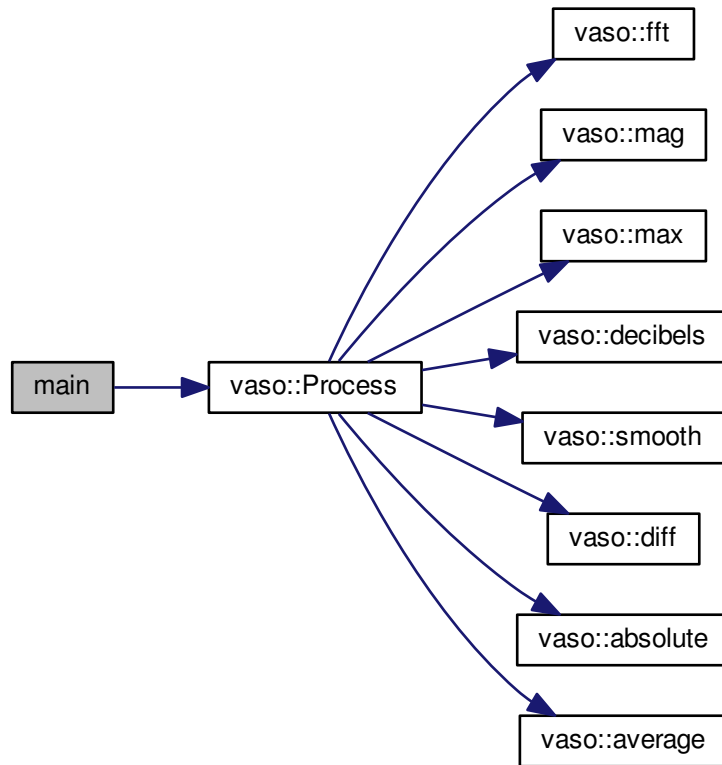
6.5.1 Function Documentation

6.5.1.1 `int main (int argc, char ** argv)`

The main program for this project. It will detect vasospasms over a period of days.

Definition at line 26 of file [main.cpp](#).

Here is the call graph for this function:



6.6 main.cpp

```

00001
00007 #include <array>
00008 #include <cstdlib>
00009 #include <iostream>
00010 #include <map>
00011 #include <pthread.h>
00012 #include <string>
00013 #include <unistd.h>
00014
00015 #include "definitions.hpp"
00016 #include "fileio.hpp"
00017 #include "process.hpp"
00018
00019 using namespace std;
00020 using namespace vaso;
00021
00026 int main(int argc, char** argv) {
00027     // generate name for patient's file
00028     string filename = ""; //PatientName();
00029
00030     // TODO: Load all of patient's parameters
00031
00032     // Record doppler audio
00033     float32 buffer[REC_COUNT][SAMPLE_COUNT];
00034
00035     for(uint8 i = 0; i < REC_COUNT; i++) {
00036         // TODO: Prompt user to press ENTER to start recording
00037
00038         int retSeek = 0; //fseek(STDIN_FILENO, 0, SEEK_END);
00039         int retRead = read(STDIN_FILENO, &buffer[i], SAMPLE_COUNT);
  
```

```

00040
00041     if(retSeek != 0 || retRead < SAMPLE_COUNT) {
00042         cerr << "An error occurred reading the doppler audio! "
00043             "The program will now exit." << endl;
00044         return ERROR;
00045     }
00046
00047     // TODO: Print message about recording stopped
00048 }
00049
00050 map<Side, DataParams> results = Process(buffer);
00051
00052 // TODO: Print results & probable diagnosis
00053
00054 // TODO: Write all results to file
00055 }

```

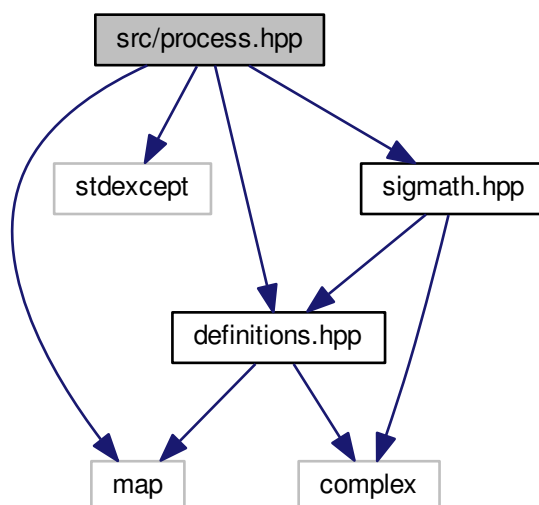
6.7 src/process.hpp File Reference

```

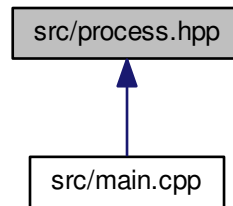
#include <map>
#include <stdexcept>
#include "definitions.hpp"
#include "sigmath.hpp"

```

Include dependency graph for process.hpp:



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



Namespaces

- [vaso](#)
contains functions related to the file I/O use in this program

Functions

- `std::map< Side, DataParams > vaso::Process (float32 data[REC_COUNT][SAMPLE_COUNT])`

6.8 process.hpp

```

00001
00007 #ifndef process_H
00008 #define process_H
00009
00010 #include <map>
00011 #include <stdexcept>
00012
00013 #include "definitions.hpp"
00014 #include "sigmath.hpp"
00015
00016 namespace vaso {
00054     std::map<Side, DataParams> Process(float32 data[REC_COUNT][
SAMPLE_COUNT]) {
00055         // just in case SAMPLE_COUNT isn't a power of two
00056         if((SAMPLE_COUNT & (SAMPLE_COUNT - 1) != 0) || SAMPLE_COUNT < 2) {
00057             throw std::invalid_argument(
00058                 "The number of samples is not a power of two!");
00059         }
00060
00061         // declare function-scoped variables
00062         uint32 freqSize = SAMPLE_COUNT / 2;
00063         cfloat32 cdata[REC_COUNT][SAMPLE_COUNT];
00064         float32 fdata[REC_COUNT][freqSize];
00065         DataParams tempParams[REC_COUNT];
00066         std::map<Side, DataParams> sideParams;
00067
00068         for(uint8 rCount = 0; rCount < REC_COUNT; rCount++) {
00069             // convert data to complex numbers for fft()
00070             for(uint32 i = 0; i < SAMPLE_COUNT; i++) {
00071                 cdata[rCount][i] = data[rCount][i];
00072             }
00073
00074             // find frequency spectrum in relative decibels
00075             fft(cdata[rCount], SAMPLE_COUNT);
00076             mag(cdata[rCount], fdata[rCount], freqSize);
00077             Maximum maximum = max(fdata[rCount], freqSize);
00078
00079             for(uint32 i = 0; i < freqSize; i++) {
00080                 fdata[rCount][i] /= maximum.value;
00081             }

```

```

00082
00083     decibels(fdata[rCount], freqSize);
00084
00085     /*
00086     * Run spectrum values through moving-average filter to smooth the
00087     * curve and make it easier to determine the derivative.
00088     */
00089     smooth(fdata[rCount], freqSize, 20);
00090
00091     /*
00092     * Find the derivative of the smoothed spectrum. Note that both this
00093     * filter and the previous are necessary to the algorithm.
00094     */
00095     diff(fdata[rCount], freqSize);
00096     smooth(fdata[rCount], freqSize, 100);
00097     absolute(fdata[rCount], freqSize);
00098
00099     // find the parameters of this specific recording
00100     uint16 offset = 1000;
00101     absolute(&fdata[rCount][offset], freqSize - offset);
00102     uint32 index = max(&fdata[rCount][offset],
00103         freqSize - offset).index;
00104     tempParams[rCount].freq = index * (float)SAMPLE_FREQ / freqSize;
00105     tempParams[rCount].noise =
00106         average(&fdata[rCount][index + 2 * offset],
00107             freqSize - 2 * offset);
00108 }
00109
00110 // calculate the parameters for each side to be returned
00111 sideParams[Side::Left] = average(&tempParams[0], REC_COUNT / 2);
00112 sideParams[Side::Right] = average(&tempParams[REC_COUNT / 2],
00113     REC_COUNT / 2);
00114 return sideParams;
00115 }
00116 }
00117
00118 #endif

```

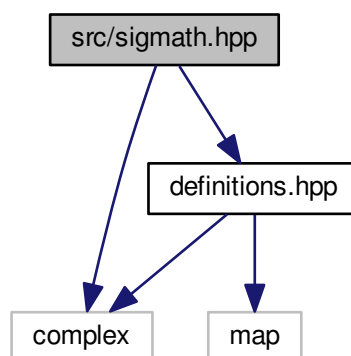
6.9 src/sigmath.hpp File Reference

```

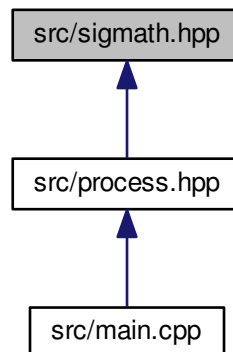
#include <complex>
#include "definitions.hpp"

```

Include dependency graph for sigmath.hpp:



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



Namespaces

- [vaso](#)

contains functions related to the file I/O use in this program

Functions

- void [vaso::absolute](#) (float32 *data, uint32 size)
- float32 [vaso::average](#) (float32 *data, uint32 size)
- DataParams [vaso::average](#) (DataParams *params, uint8 size)
- void [vaso::average](#) (float32 *data, float32 *avg, uint8 count, uint32 size)
- void [vaso::decibels](#) (float32 *data, uint32 size)
- void [vaso::diff](#) (float32 *data, uint32 size)
- void [vaso::fft](#) (cfloat32 *data, uint32 size)
- void [vaso::mag](#) (cfloat32 *orig, float32 *newmags, uint32 size)
- Maximum [vaso::max](#) (float32 *data, uint32 size)
- void [vaso::smooth](#) (float32 *data, uint32 size, uint16 order)

6.10 sigmath.hpp

```

00001
00008 #ifndef sigmath_H
00009 #define sigmath_H
00010
00011 #include <complex>
00012
00013 #include "definitions.hpp"
00014
00015 namespace vaso {
00016     // PROTOTYPES
00017
00026     void absolute(float32* data, uint32 size);
00027
00037     float32 average(float32* data, uint32 size);
00038
00049     DataParams average(DataParams* params, uint8 size);
00050
00067     void average(float32* data, float32* avg, uint8 count,
uint32 size);
  
```

```

00068
00080 void decibels(float32* data, uint32 size);
00081
00090 void diff(float32* data, uint32 size);
00091
00103 void fft(cfloat32* data, uint32 size);
00104
00114 void mag(cfloat32* orig, float32* newmags, uint32 size);
00115
00125 Maximum max(float32* data, uint32 size);
00126
00137 void smooth(float32* data, uint32 size, uint16 order);
00138
00139 // DEFINITIONS
00140
00141 void absolute(float32* data, uint32 size) {
00142 }
00143
00144 float32 average(float32* data, uint32 size) {
00145 }
00146
00147 DataParams average(DataParams* params, uint8 size) {
00148 }
00149
00150 void average(float32* data, float32* avg, uint8 count,
00151 uint32 size) {
00152     // data is an array. Access like so: data[index]
00153 }
00154
00155 void decibels(float32* data, uint32 size) {
00156     for(uint32 i = 0; i < size; i++) {
00157         data[i] = 20 * log10(data[i]);
00158     }
00159 }
00160
00161 void diff(float32* data, uint32 size) {
00162 }
00163
00164 void fft(cfloat32* data, uint32 size) {
00165     // DFT
00166     uint32 k = size;
00167     uint32 n;
00168     float32 thetaT = M_PI / size;
00169     cfloat32 phiT(cos(thetaT), sin(thetaT));
00170     cfloat32 T;
00171
00172     while(k > 1) {
00173         n = k;
00174         k >>= 1;
00175         phiT = phiT * phiT;
00176         T = 1.0L;
00177
00178         for(uint32 l = 0; l < k; l++) {
00179             for(uint32 a = l; a < size; a += n) {
00180                 uint32 b = a + k;
00181                 cfloat32 t = data[a] - data[b];
00182                 data[a] += data[b];
00183                 data[b] = t * T;
00184             }
00185
00186             T *= phiT;
00187         }
00188     }
00189
00190 // Decimate
00191 uint32 m = (uint32)log2(size);
00192
00193 for(uint32 a = 0; a < size; a++) {
00194     uint32 b = a;
00195
00196     // Reverse bits
00197     b = ((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1);
00198     b = ((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2);
00199     b = ((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4);
00200     b = ((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8);
00201     b = ((b >> 16) | (b << 16)) >> (32 - m);
00202
00203     if (b > a)
00204     {
00205         cfloat32 t = data[a];
00206         data[a] = data[b];
00207         data[b] = t;
00208     }
00209 }
00210
00211

```

```

00212     }
00213 }
00214
00215 void mag(cfloat32* orig, float32* newmags, uint32 size) {
00216 }
00217 }
00218
00219 Maximum max(float32* data, uint32 size) {
00220 }
00221 }
00222
00223 void smooth(float32* data, uint32 size, uint16 order) {
00224     float32 coeff = 1 / (float32)order;
00225     float32 temp[size];
00226
00227     for(uint32 i = 0; i < size; i++) {
00228         temp[i] = 0;
00229
00230         for(uint16 j = 0; j < order && j <= i; j++) {
00231             temp[i] += data[i - j];
00232         }
00233
00234         temp[i] *= coeff;
00235     }
00236 }
00237 }
00238
00239 #endif

```

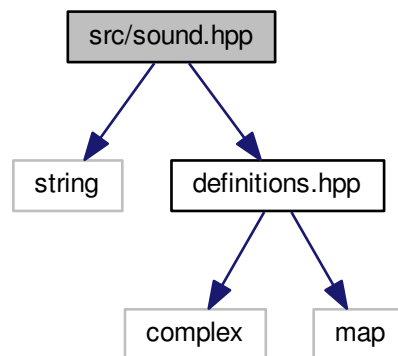
6.11 src/sound.hpp File Reference

```

#include <string>
#include "definitions.hpp"

```

Include dependency graph for sound.hpp:



Namespaces

- [vaso](#)
contains functions related to the file I/O use in this program

Functions

- void [vaso::play](#) (auto filename)

6.12 sound.hpp

```
00001
00006 #ifndef sound_H
00007 #define sound_H
00008
00009 #include <string>
00010
00011 #include "definitions.hpp"
00012
00013 namespace vaso {
00019     void play(auto filename) {
00020
00021     }
00022 }
00023
00024 #endif
```

Index

- absolute
 - vaso, [8](#)
- average
 - vaso, [8, 9](#)
- decibels
 - vaso, [10](#)
- diff
 - vaso, [10](#)
- fft
 - vaso, [10](#)
- index
 - Maximum, [18](#)
- Left
 - vaso, [8](#)
- mag
 - vaso, [11](#)
- max
 - vaso, [11](#)
- Maximum, [17](#)
 - index, [18](#)
 - value, [18](#)
- play
 - vaso, [12](#)
- Process
 - vaso, [12](#)
- Right
 - vaso, [8](#)
- Side
 - vaso, [8](#)
- smooth
 - vaso, [14](#)
- value
 - Maximum, [18](#)
- vaso, [7](#)
 - absolute, [8](#)
 - average, [8, 9](#)
 - decibels, [10](#)
 - diff, [10](#)
 - fft, [10](#)
 - Left, [8](#)
 - mag, [11](#)
 - max, [11](#)
 - play, [12](#)
 - Process, [12](#)
 - Right, [8](#)
 - Side, [8](#)
 - smooth, [14](#)