Andrew and Nick's Project

Generated by Doxygen 1.8.8

Mon Apr 18 2016 22:27:19

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 . . . 8

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 average . . . 10

4.1.3.6 decibels . . . 10

4.1.3.7 diff . . . 10

4.1.3.8 fft . . . 11

4.1.3.9 mag . . . 11

4.1.3.10 max . . . 12

4.1.3.11 PatientName . . . 12

4.1.3.12 play . . . 13

4.1.3.13 Process . . . 13

4.1.3.14 process . . . 14

4.1.3.15 ReadParams . . . 15

4.1.3.16 smooth . . . 16

4.1.3.17 WriteParams . . . 16

4.1.4 Variable Documentation . . . 16
4.1.4.1 CSV_HEADER . . . 16
4.1.4.2 PATIENT_PATH . . . 16

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 . . . 22
6.1.1.1 ENUM . . . 22
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 . . . 23
6.1.2.4 float64 . . . 23
6.1.2.5 sint16 . . . 23
6.1.2.6 sint32 . . . 23
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
6.3.1 Detailed Description . . . 26
6.4 fileio.hpp . . . 26
6.5 src/main.cpp File Reference . . . 28
6.5.1 Detailed Description . . . 28
6.5.2 Function Documentation . . . 28
6.5.2.1 main . . . 28
6.6 main.cpp . . . 29
6.7 src/patient_name_test.cpp File Reference . . . 30
6.7.1 Function Documentation . . . 30
6.7.1.1 main . . . 30
6.8 patient_name_test.cpp . . . 31
6.9 src/process.hpp File Reference . . . 31
6.10 process.hpp . . . 32
6.11 src/process_test.cpp File Reference . . . 34
6.11.1 Detailed Description . . . 35
6.11.2 Macro Definition Documentation . . . 35
6.11.2.1 COUNT . . . 35
6.11.3 Function Documentation . . . 35
6.11.3.1 main . . . 35
6.12 process_test.cpp . . . 36
6.13 src/sigmath.hpp File Reference . . . 37
6.13.1 Detailed Description . . . 38
6.14 sigmath.hpp . . . 38
6.15 src/sigmath_test.cpp File Reference . . . 40
6.15.1 Function Documentation . . . 41
6.15.1.1 main . . . 41
6.16 sigmath_test.cpp . . . 41
6.17 src/sound.hpp File Reference . . . 41
6.18 sound.hpp . . . 42

Index **43**

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

vaso
Function()s related to the program's threaded processing of audio data 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
    Functions related to the file I/O use in this program 25
src/main.cpp
    Main program 28
src/patient_name_test.cpp 30
src/process.hpp 31
src/process_test.cpp 34
src/sigmath.hpp
    Functions necessary to perform the mathematical operations required by this program 37
src/sigmath_test.cpp 40
src/sound.hpp 41

Chapter 4

Namespace Documentation

4.1 vaso Namespace Reference

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

Enumerations

- enum Side { Side::Left, Side::Right }

Functions

- std::string PatientName ()
- std::map< Side, DataParams > ReadParams (auto filename)
- std::string WriteParams (DataParams params, auto filename)
- std::map< Side, DataParams > Process (float32 **data)
- DataParams process (float32 *data, uint32 size, float32 samplingRate)
- 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 average (float32 **data, float32 *avg, uint8 count, uint32 size)
- void play (auto filename)

Variables

- const std::string CSV_HEADER = "Time,Side,Frequency,Noise Level"
- const std::string PATIENT_PATH = "/home/pi/patients/"

4.1.1 Detailed Description

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

contains the function(s) relating to sound

This namespace contains all code related to this project.

Author

Samuel Andrew Wisner, awisner94@gmail.com

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*)

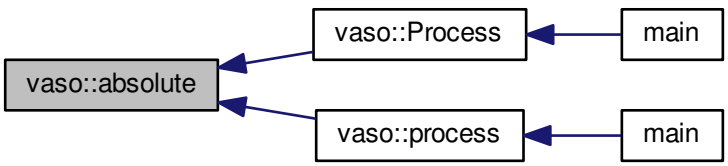
Ensures all elements in an array are positive. Note that this function replaces array elements if necessary. It does not populate a new array.

Parameters

data	the array whose elements must all be positive
size	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

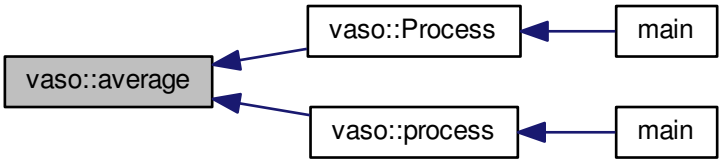
data	the array from which to compute the average
size	the number of elements in the data array

Returns

the computed average

Definition at line 147 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

params	the DataParams array
size	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 158 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

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

size	the number of elements in the second dimension of data

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

Definition at line 173 of file sigmath.hpp.

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

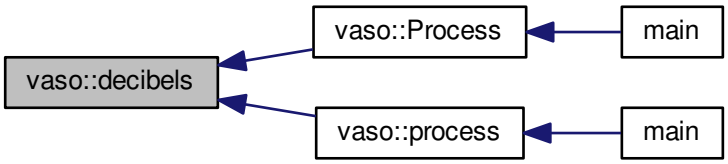
Converts an array of floats to "power decibels", i.e., x[n] = 20*log10(x[n]). The decibel values are written to the same array that contained the values to be converted. In other words, this function should perform an in-place, element-wise conversion.

Parameters

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

Definition at line 189 of file sigmath.hpp.

Here is the caller graph for this function:



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

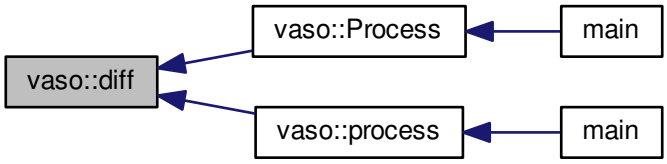
Computes the left-handed first derivative of a discrete signal. The first element will be 0.

Parameters

data	an array containing the discrete signal data
size	the number of elements in data

Definition at line 195 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.

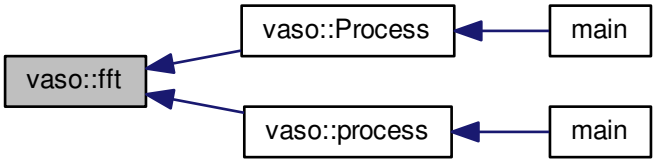
This code is based on code from http://rosettacode.org/wiki/Fast_Fourier_transform#C.↩2B.2B.

Parameters

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

Definition at line 208 of file sigmath.hpp.

Here is the caller graph for this function:



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

Computes the magitude of an array of complex numbers.

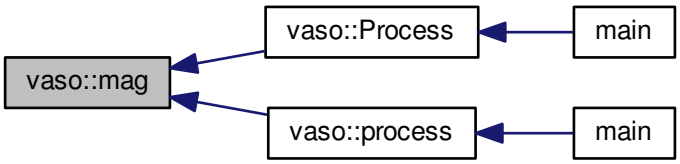
Parameters

orig	the array of complex numbers
newmags	an array to which the magitudes are to be written

size	the number of elements in orig and newmags

Definition at line 256 of file sigmath.hpp.

Here is the caller graph for this function:



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

Finds the maximum value in an array.

Parameters

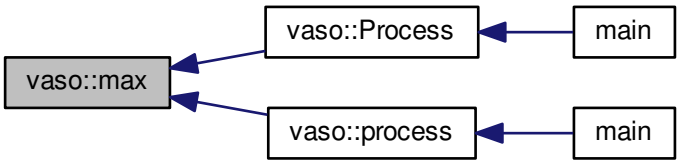
data	the array whose maximum value is to be found
uint32	size the number of elements in the data array

Returns

the maximum value and its index in a Maximum structure

Definition at line 267 of file sigmath.hpp.

Here is the caller graph for this function:



4.1.3.11 std::string vaso::PatientName ( )

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

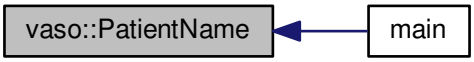
Must warn a user if the patient folder does not already exist in order to prevent missaving data.

Returns

the file under which all patient data is saved

Definition at line 37 of file fileio.hpp.

Here is the caller graph for this function:



4.1.3.12 void vaso::play (auto *filename*)

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

Parameters

filename	the absolute or relative path to the WAVE file

Definition at line 19 of file sound.hpp.

4.1.3.13 std::map<Side, DataParams> vaso::Process (float32 ** *data*)

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 prioprietary 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.

Parameters

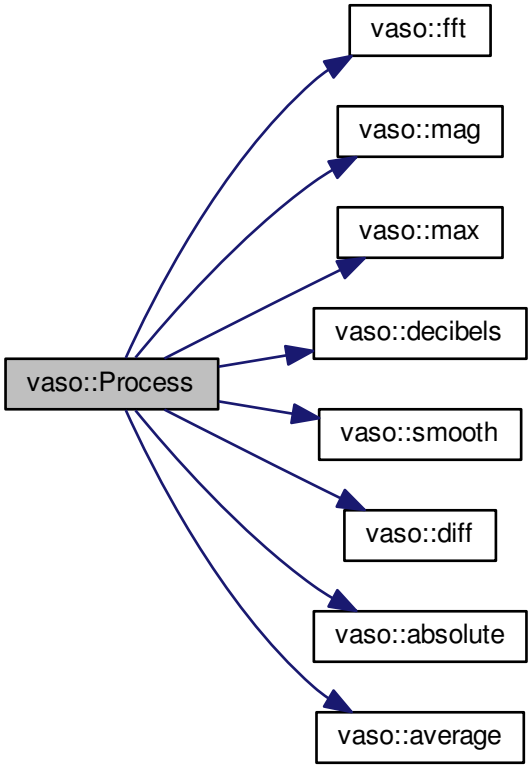
data	two-dimensional array (first dimension whole recordings, second dimension samples in a recording) that will contain all recorded audio
REC_COUNT	the number of recordings (left and right together) to be made
SAMPLE_COUNT	the number of samples in each recording. MUST be a power of two.
SAMPLE_FREQ	the sampling frequency in Hz or Samples/second

Returns

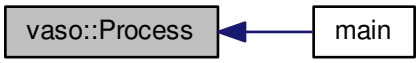
a map of the averaged left- and right-side parameters in DataParams structures

Definition at line 54 of file process.hpp.

Here is the call graph for this function:



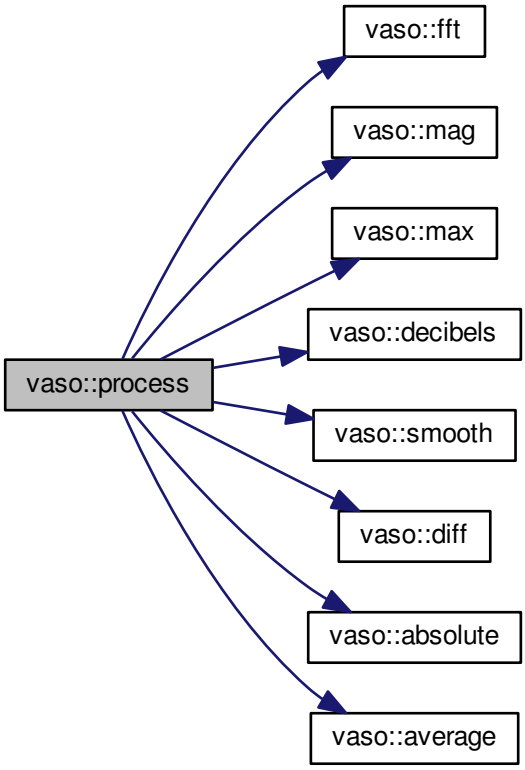
Here is the caller graph for this function:



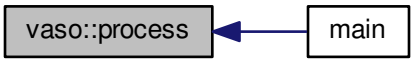
4.1.3.14 DataParams vaso::process (float32 * *data,* uint32 *size,* float32 *samplingRate*)

Definition at line 117 of file process.hpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.15 std::map<Side, DataParams> vaso::ReadParams (auto *filename*)

Reads the previously computated parameters found in the specified file.

Parameters

filename	the absolute or relative path to the file containing the patient data to read

Returns

the patient parameters read

Definition at line 127 of file fileio.hpp.

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

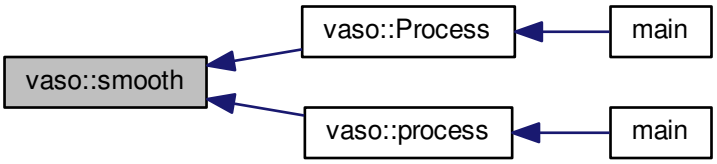
Applies an nth-order moving-average filter to a discrete signal.

Parameters

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

Definition at line 285 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

params	

Definition at line 153 of file fileio.hpp.

4.1.4 Variable Documentation

4.1.4.1 const std::string vaso::CSV_HEADER = "Time,Side,Frequency,Noise Level"

First line of CSV data file, which declares columns.

Definition at line 20 of file fileio.hpp.

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

Absolute path to the folder containing the patients' data

Definition at line 25 of file fileio.hpp.

Chapter 5

Class Documentation

5.1 DataParams Struct Reference

`#include <definitions.hpp>`

Public Attributes

- float32 freq = 0
- float32 noise = 0

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 = 0

Definition at line 45 of file definitions.hpp.

5.1.2.2 float32 DataParams::noise = 0

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 = 0
- uint32 index = 0

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 = 0

Definition at line 55 of file definitions.hpp.

5.2.2.2 float32 Maximum::value = 0

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

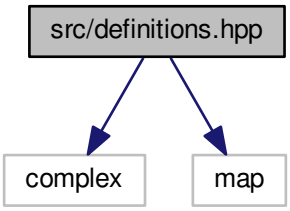
Chapter 6

File Documentation

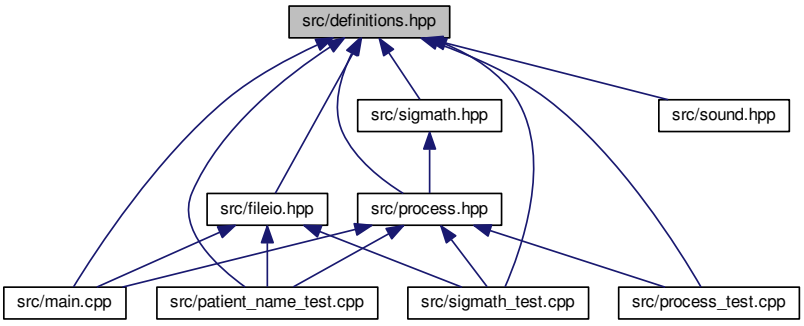
6.1 src/definitions.hpp File Reference

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

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 6
- #define SAMPLE_COUNT 262144
- #define SAMPLE_FREQ 44100
- #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 6

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 44100

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 6
00016 #define SAMPLE_COUNT 262144
00017 #define SAMPLE_FREQ 44100
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 = 0;
00046     float32 noise = 0;
00047 } DataParams;
00048
00053 typedef struct {
00054     float32 value = 0;
00055     uint32 index = 0;
00056 } Maximum;
```

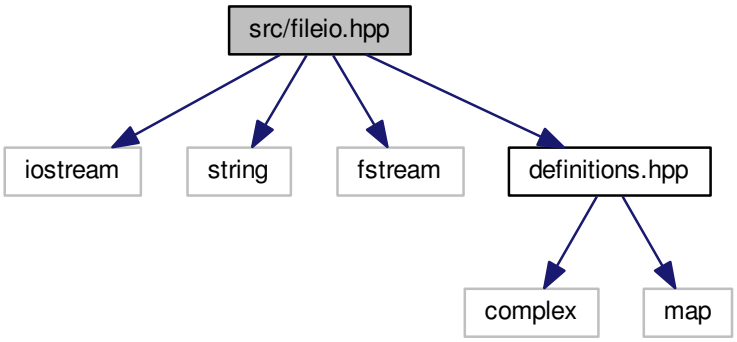
```
00057
00061 namespace vaso {
00065     enum class Side { Left, Right };
00066 }
00067
00072 typedef struct {
00073     float32** data;
00074     uint8 recCount;
00075     uint32 sampleCount;
00076     uint32 sampleFreq;
00077     uint8* counter;
00078     std::map<vaso::Side, DataParams> results;
00079 } ThreadParams;
00080
00081 #endif
```

6.3 src/fileio.hpp File Reference

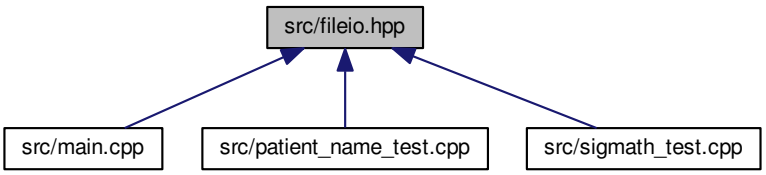
contains functions related to the file I/O use in this program

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

Include dependency graph for fileio.hpp:



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



Namespaces

- vaso

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

Functions

- std::string vaso::PatientName ()
- std::map< Side, DataParams > vaso::ReadParams (auto filename)
- std::string vaso::WriteParams (DataParams params, auto filename)

Variables

- const std::string vaso::CSV_HEADER = "Time,Side,Frequency,Noise Level"
- const std::string vaso::PATIENT_PATH = "/home/pi/patients/"

6.3.1 Detailed Description

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

Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file fileio.hpp.

6.4 fileio.hpp

```
00001
00007 #ifndef fileio_H
00008 #define fileio_H
00009
00010 #include <iostream>
00011 #include <string>
00012 #include <fstream>
00013
00014 #include "definitions.hpp"
00015
00016 namespace vaso {
00020     const std::string CSV_HEADER = "Time,Side,Frequency,Noise Level";
00021
00025     const std::string PATIENT_PATH = "/home/pi/patients/";
00026
00037     std::string PatientName() {
00038         std::string fname = "";
00039         std::string mname = "";
00040         std::string lname = "";
00041         std::string patfil = "";
00042         std::string patientname = "";
00043         uint32 track1 = 0;
00044         uint32 track2 = 0;
00045
00046         do {
00047             std::cout << "Please enter the patients name." << std::endl;
00048             std::cout << "First name: ";
00049             std::cin >> fname;
00050             std::cout << "Middle name: ";
00051             std::cin >> mname;
00052             std::cout << "Last name: ";
00053             std::cin >> lname;
00054
00055             // creates new std::string with path to patient file
00056             patientname = PATIENT_PATH + lname + ", " + fname
00057                 + " " + mname + ".csv";
00058
00059             // prints out patientname. shows user the path to the patient file
00060             std::cout << patientname << std::endl << std::endl;
```

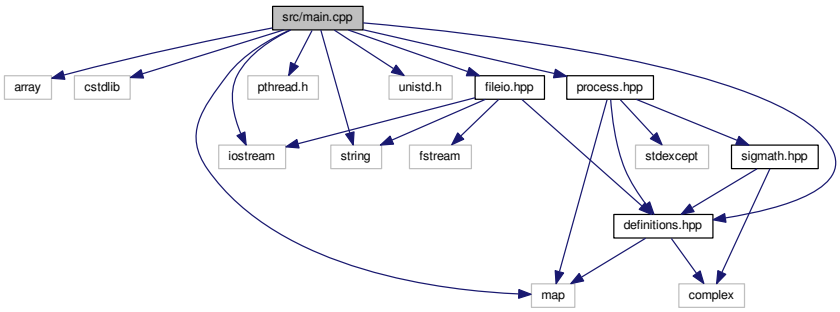
```
00061             std::ifstream file(patientname.c_str());
00062
00063             /*
00064              * Compares patientname to existing files and lets user know
00065              * if the file does not exist.
00066              */
00067             if (!file.good()) {
00068                 /*
00069                  * Do while statement to continue asking user about the file
00070                  * if their input is not acceptable
00071                  */
00072                 do {
00073                     std::cout << "Patient file does not exist, would you like "
00074                         "to create file or re-enter their name?" << std::endl;
00075                     std::cout << "  *Type 'create' and press enter key "
00076                         "to create the patient file." << std::endl;
00077                     std::cout << "  *Type 'reenter' and press enter key "
00078                         "to re-enter the patients name." << std::endl;
00079                     std::cout << std::endl;
00080                     std::cin >> patfil;
00081
00082                     /*
00083                      * patfil equals create, track1 and 2 will increase
00084                      * escaping both do while loops
00085                      */
00086                     if(patfil == "create") {
00087                         track1 = 1;
00088                         track2 = 1;
00089                         file.open(patientname);
00090                         file.close();
00091                     }
00092
00093                     /*
00094                     *patfil equals renter, track1 will remain zero allowing
00095                     *user to reenter the patient name.
00096                     */
00097                     else if(patfil == "reenter") {
00098                         track1 = 0;
00099                         track2 = 1;
00100                     }
00101
00102                     /*
00103                     *The users input was neither create or reenter. User
00104                     *must enter patient name again.
00105                     */
00106                     else {
00107                         std::cout << std::endl;
00108                         std::cout << "Your input is not acceptable." << std::endl;
00109                         std::cout << std::endl;
00110                     }
00111                 }while(track2 == 0);
00112             }
00113         } while (track1 = 0);
00114
00115         return patientname; //returns the path to the patient file
00116     }
00117
00127     std::map<Side, DataParams> ReadParams(auto filename) {
00128         DataParams par;
00129         std::ifstream file(filename.c_str());
00130         std::string line;
00131         //if statement which uses ifstream function to open patient file (filename)
00132         if(file.is_open()) {
00133             std::getline(file, line);
00134         }
00135
00136         else {
00137             std::cout << "The patient file could not be opened." << std::endl;
00138         }
00139
00140         std::map<Side, DataParams> myMap;
00141         DataParams myParams;
00142         myMap[Side::Left] = myParams;
00143
00144         std::par = line;
00145         return par;
00146     }
00147
00153     std::string WriteParams(DataParams params, auto filename) {
00154
00155     }
00156 }
00157
00158 #endif
00159
```

6.5 src/main.cpp File Reference

contains the main program

```
#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 Detailed Description

contains the main program

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Definition in file main.cpp.

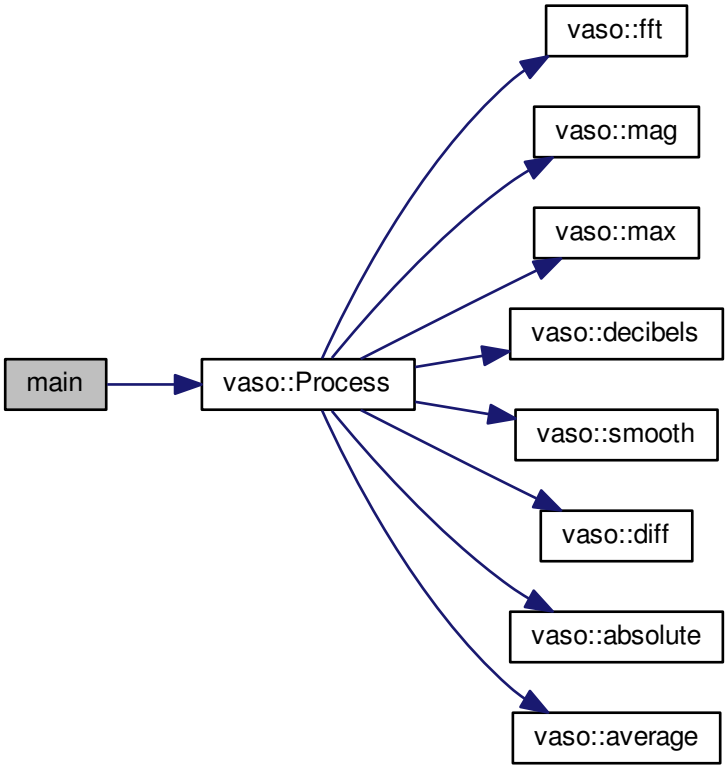
6.5.2 Function Documentation

6.5.2.1 int main (int *argc,* char ** *argv*)

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

Definition at line 27 of file main.cpp.

Here is the call graph for this function:



6.6 main.cpp

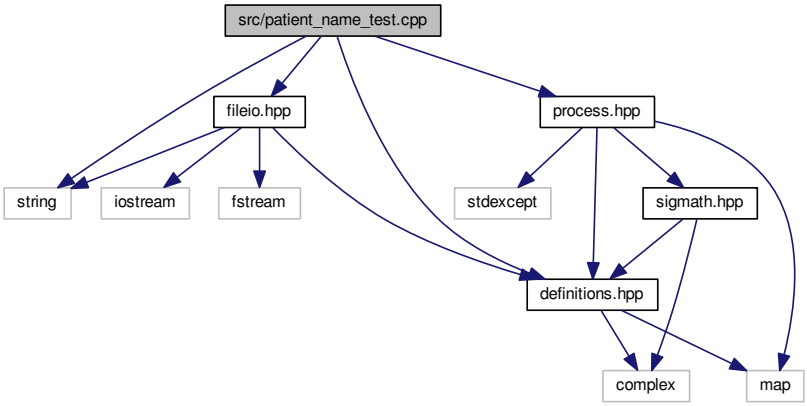
```
00001
00008 #include <array>
00009 #include <cstdlib>
00010 #include <iostream>
00011 #include <map>
00012 #include <pthread.h>
00013 #include <string>
00014 #include <unistd.h>
00015
00016 #include "definitions.hpp"
00017 #include "fileio.hpp"
00018 #include "process.hpp"
00019
00020 using namespace std;
00021 using namespace vaso;
00022
00027 int main(int argc, char** argv) {
00028     // generate name for patient's file
00029     string filename = "";//PatientName();
00030
00031     // TODO: Load all of patient's parameters
00032
00033     // Record doppler audio
00034     float32* buffer[REC_COUNT];
00035
00036     for(uint8 i = 0; i < REC_COUNT; i++) {
00037         buffer[i] = (float32*)malloc(SAMPLE_COUNT * sizeof(
      float32));
00038     }
00039
```

```
00040     for(uint8 i = 0; i < REC_COUNT; i++) {
00041         // TODO: Prompt user to press ENTER to start recording
00042
00043         int retSeek = fseek(STDIN_FILENO, 0, SEEK_END);
00044         int retRead = read(STDIN_FILENO, &buffer[i], SAMPLE_COUNT);
00045
00046         if(retSeek != 0 || retRead < SAMPLE_COUNT) {
00047             cerr << "An error occurred reading the doppler audio! "
00048                 "The program will now exit." << endl;
00049             return ERROR;
00050         }
00051
00052         // TODO: Print message about recording stopped
00053     }
00054
00055     map<Side, DataParams> results = Process(buffer);
00056
00057     // TODO: Print results & probable diagnosis
00058
00059     // TODO: Write all results to file
00060 }
```

6.7 src/patient_name_test.cpp File Reference

```
#include <string>
#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"
```

Include dependency graph for patient_name_test.cpp:



Functions

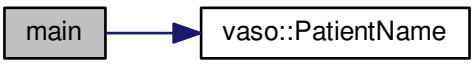
- int main (int argc, char **argv)

6.7.1 Function Documentation

6.7.1.1 int main (int *argc*, char ** *argv*)

Definition at line 20 of file patient_name_test.cpp.

Here is the call graph for this function:



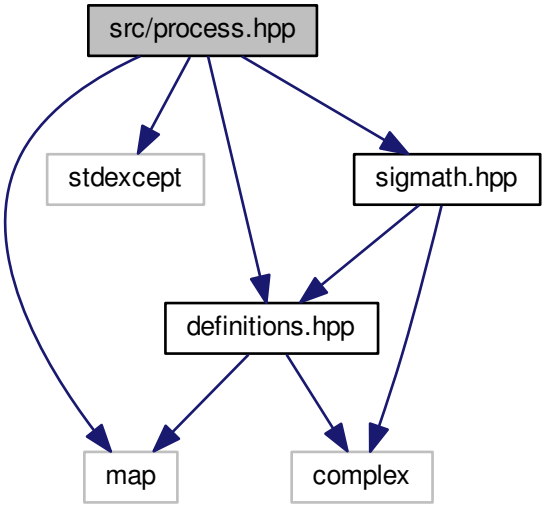
6.8 patient_name_test.cpp

```
00001
00007 #include <string>
00008
00009 #include "definitions.hpp"
00010 #include "fileio.hpp"
00011 #include "process.hpp"
00012
00013 using namespace std;
00014 using namespace vaso;
00015
00020 int main(int argc, char** argv) {
00021     string filename = PatientName();
00022     cout << filename;
00023 }
```

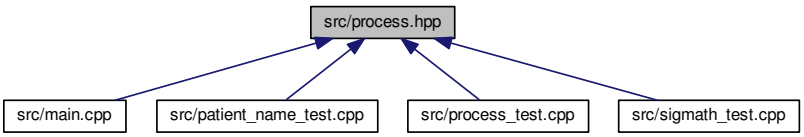
6.9 src/process.hpp File Reference

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

Functions

- std::map< Side, DataParams > vaso::Process (float32 **data)
- DataParams vaso::process (float32 *data, uint32 size, float32 samplingRate)

6.10 process.hpp

```
00001
00007 #ifndef process_H
00008 #define process_H
00009
00010 #include <map>
00011 #include <stdexcept>
```

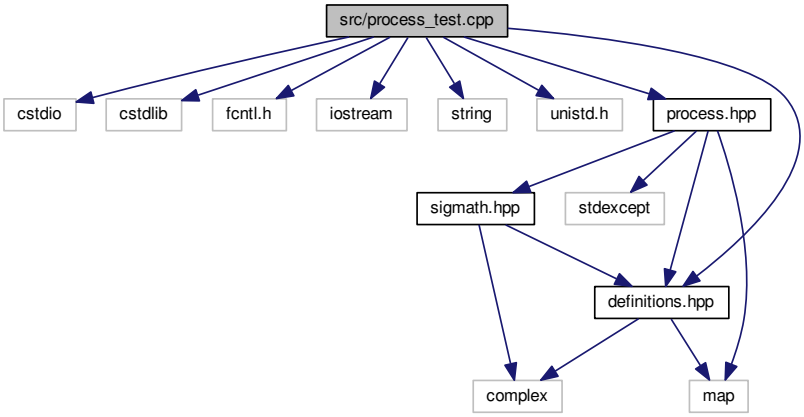
```
00012
00013 #include "definitions.hpp"
00014 #include "sigmath.hpp"
00015
00016 namespace vaso {
00054     std::map<Side, DataParams> Process(float32** data) {
00055         // just in case SAMPLE_COUNT isn't a power of two
00056         if((SAMPLE_COUNT & (SAMPLE_COUNT - 1) != 0) ||
      SAMPLE_COUNT < 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. Bote 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     DataParams process(float32* data, uint32 size,
      float32 samplingRate) {
00118         // just in case SAMPLE_COUNT isn't a power of two
00119         if((size & (size - 1) != 0) || size < 2) {
00120             throw std::invalid_argument(
00121                     "The number of samples is not a power of two!");
00122         }
00123
00124         // declare function-scoped variables
00125         uint32 freqSize = size / 2;
00126         cfloat32* cdata = (cfloat32*)std::malloc(size * sizeof(
      cfloat32));
00127         float32* fdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00128         float32* origdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00129
00130         // convert data to complex numbers for fft()
```

```
00131         for(uint32 i = 0; i < size; i++) {
00132             cdata[i] = data[i];
00133         }
00134 
00135         // find frequency spectrum in relative decibels
00136         fft(cdata, size);
00137         mag(cdata, fdata, freqSize);
00138         Maximum maximum = max(fdata, freqSize);
00139 
00140         for(uint32 i = 0; i < freqSize; i++) {
00141             fdata[i] /= maximum.value;
00142         }
00143 
00144         decibels(fdata, freqSize);
00145 
00146         for(uint32 i = 0; i < freqSize; i++) {
00147             origdata[i] = fdata[i];
00148         }
00149 
00150         /*
00151          * Run spectrum values through moving-average filter to smooth the
00152          * curve and make it easier to determine the derivative.
00153          */
00154         smooth(fdata, freqSize, 20);
00155 
00156         /*
00157          * Find the derivative of the smoothed spectrum. Bote that both this
00158          * filter and the previous are necessary to the algorithm.
00159          */
00160         diff(fdata, freqSize);
00161         smooth(fdata, freqSize, 100);
00162         absolute(fdata, freqSize);
00163 
00164         // find the parameters of this specific recording
00165         uint16 offset = 1000;
00166         absolute(&fdata[offset], freqSize - offset);
00167         maximum = max(&fdata[offset], freqSize - offset);
00168         uint32 index = maximum.index + offset;
00169         DataParams params;
00170         params.freq = index * (float)SAMPLE_FREQ / freqSize / 2;
00171         params.noise = average(&origdata[index + offset],
00172                 freqSize - offset - index);
00173 
00174         free(cdata);
00175         free(fdata);
00176         return params;
00177 
00178     }
00179 }
00180 
00181 #endif
```

6.11 src/process_test.cpp File Reference

```
#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#include <iostream>
#include <string>
#include <unistd.h>
#include "definitions.hpp"
#include "process.hpp"
```

Include dependency graph for process_test.cpp:



Macros

- #define COUNT 131072

Functions

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

6.11.1 Detailed Description

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Definition in file process_test.cpp.

6.11.2 Macro Definition Documentation

6.11.2.1 #define COUNT 131072

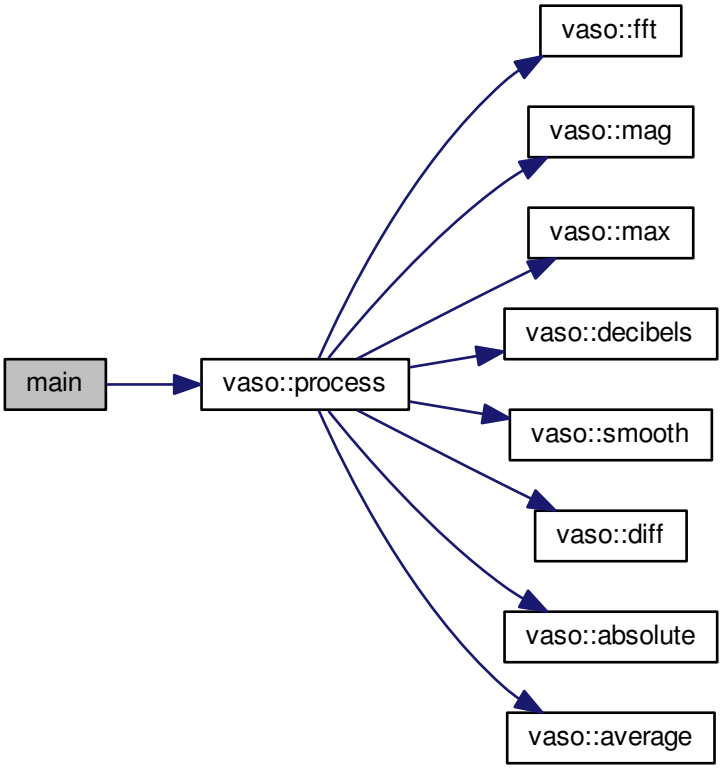
Definition at line 18 of file process_test.cpp.

6.11.3 Function Documentation

6.11.3.1 int main (int *argc*, char ** *argv*)

Definition at line 26 of file process_test.cpp.

Here is the call graph for this function:



6.12 process_test.cpp

```
00001
00008 #include <cstdio>
00009 #include <cstdlib>
00010 #include <fcntl.h>
00011 #include <iostream>
00012 #include <string>
00013 #include <unistd.h>
00014
00015 #include "definitions.hpp"
00016 #include "process.hpp"
00017
00018 #define COUNT 131072
00019
00020 using namespace std;
00021 using namespace vaso;
00022
00026 int main(int argc, char** argv) {
00027     int file = open("/home/pi/vaso/etc/audio/test.raw", O_RDONLY);
00028
00029     if(file < 0) {
00030         cerr << "File unreadable!" << endl;
00031         return -1;
00032     }
00033
00034     float32* buffer = (float32*)malloc(COUNT * sizeof(float32));
00035     int charRead = read(file, buffer, COUNT * sizeof(float32));
00036
00037     if(charRead < COUNT) {
00038         cerr << "Too few bytes read!" << endl;
00039         return -1;
```

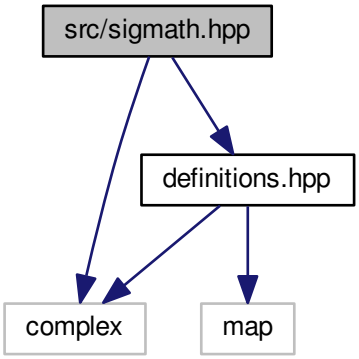
```
00040     }
00041
00042     close(file);
00043
00044     DataParams params = process(buffer, COUNT, SAMPLE_FREQ);
00045     free(buffer);
00046     cout << "Cutoff: " << params.freq << endl;
00047     cout << "Noise: " << params.noise << endl;
00048 }
```

6.13 src/sigmath.hpp File Reference

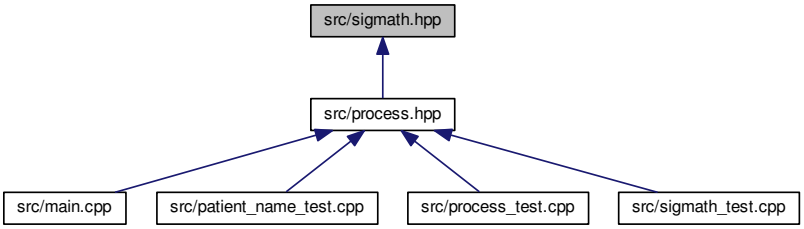
contains the functions necessary to perform the mathematical operations required by this program

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

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

6.13.1 Detailed Description

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

Author

Samuel Andrew Wisner, awisner94@gmail.com
Nicholas K. Nolan

Definition in file sigmath.hpp.

6.14 sigmath.hpp

```
00001
00009 #ifndef sigmath_H
00010 #define sigmath_H
00011
00012 #include <complex>
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         for(uint32 i = 0; i < size; i++) {
00143             data[i] = fabsf(data[i]);
00144         }
00145     }
00146
00147     float32 average(float32* data, uint32 size) {
00148         float32 ave;
00149
00150         for(uint32 i = 0; i < size; i++) {
```

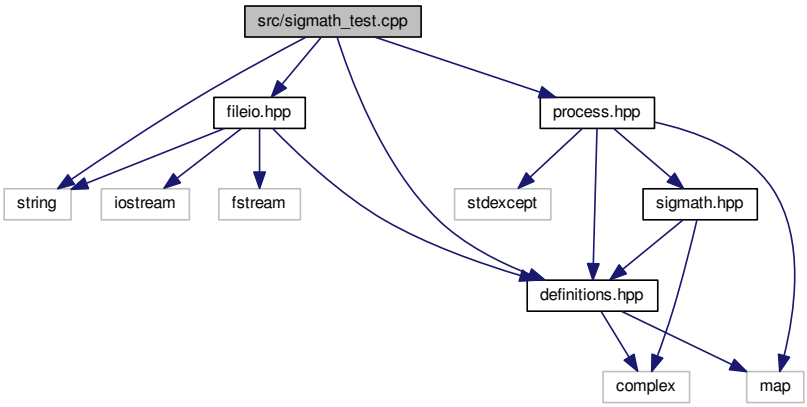
```
00151             ave += data[i];
00152         }
00153
00154         ave = ave / size;
00155         return ave;
00156     }
00157
00158     DataParams average(DataParams* params, uint8 size) {
00159         DataParams ave;
00160
00161         for(uint8 i = 0; i < size; i++) {
00162             //freq is an attribute. this is how to add structure attributes
00163             ave.freq += params[i].freq;
00164
00165             ave.noise += params[i].noise;
00166         }
00167
00168         ave.freq /= size;
00169         ave.noise /= size;
00170         return ave;
00171     }
00172
00173     void average(float32** data, float32* avg, uint8 count,
    uint32 size) {
00174         // data is an array. Access like so: data[index]
00175         //loop for the number of "columns" in the array
00176         for(uint32 e = 0; e < size; e++) {
00177             float32 c = 0;
00178
00179             //loop for the number of "rows" in the array (in case > 3)
00180             for(uint32 r = 0; r < count; r++) {
00181                 c += data [r][e];       //adds values in each row for column e
00182             }
00183
00184             c = c / count;      //averages the values
00185             avg[e] = c;     //saves averaged value into avg array
00186         }
00187     }
00188
00189     void decibels(float32* data, uint32 size) {
00190         for(uint32 i = 0; i < size; i++) {
00191             data[i] = 20 * log10(data[i]);
00192         }
00193     }
00194
00195     void diff(float32* data, uint32 size) {
00196         float32 temp[size];
00197         temp[0] = 0;
00198
00199         for(uint32 i = 1; i < size; i++) {
00200             temp[i] = data[i] - data[i-1];
00201         }
00202
00203         for(uint32 i = 0; i < size; i++) {
00204             data[i] = temp[i];
00205         }
00206     }
00207
00208     void fft(cfloat32* data, uint32 size) {
00209         // DFT
00210         uint32 k = size;
00211         uint32 n;
00212         float32 thetaT = M_PI / size;
00213         cfloat32 phiT(cos(thetaT), sin(thetaT));
00214         cfloat32 T;
00215
00216         while(k > 1) {
00217             n = k;
00218             k >>= 1;
00219             phiT = phiT * phiT;
00220             T = 1.0L;
00221
00222             for(uint32 l = 0; l < k; l++) {
00223                 for(uint32 a = l; a < size; a += n) {
00224                     uint32 b = a + k;
00225                     cfloat32 t = data[a] - data[b];
00226                     data[a] += data[b];
00227                     data[b] = t * T;
00228                 }
00229
00230                 T *= phiT;
00231             }
00232         }
00233
00234         // Decimate
00235         uint32 m = (uint32)log2(size);
00236
```

```
00237         for(uint32 a = 0; a < size; a++) {
00238             uint32 b = a;
00239 
00240             // Reverse bits
00241             b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1));
00242             b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
00243             b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
00244             b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
00245             b = ((b >> 16) | (b << 16)) >> (32 - m);
00246 
00247             if (b > a)
00248             {
00249                 cfloat32 t = data[a];
00250                 data[a] = data[b];
00251                 data[b] = t;
00252             }
00253         }
00254     }
00255 
00256     void mag(cfloat32* orig, float32* newmags, uint32 size) {
00257         //loop to run throught the length of array orig
00258         for(uint32 n = 0; n < size; n++) {
00259             /*
00260              * abs should calculate the magnitude of complex array elements.
00261              * saves to new array
00262              */
00263             newmags[n] = std::abs(orig[n]);
00264         }
00265     }
00266 
00267     Maximum max(float32* data, uint32 size) {
00268         Maximum m;
00269 
00270         //loop to run through the length of array data
00271         for (uint32 i = 0; i < size; i++) {
00272             /*
00273              * when value at data[i] is above max.value,
00274              * sets max.value equal to data[i] and max.index equal to i
00275              */
00276             if (data[i] > m.value) {
00277                 m.value = data[i];
00278                 m.index = i;
00279             }
00280         }
00281 
00282         return m;
00283     }
00284 
00285     void smooth(float32* data, uint32 size, uint16 order) {
00286         float32 coeff = 1 / (float32)order;
00287         float32 temp[size];
00288 
00289         for(uint32 i = 0; i < size; i++) {
00290             temp[i] = 0;
00291 
00292             for(uint16 j = 0; j < order && j <= i; j++) {
00293                 temp[i] += data[i - j];
00294             }
00295 
00296             temp[i] *= coeff;
00297         }
00298 
00299         for(uint32 i = 0; i < size; i++) {
00300             data[i] = temp[i];
00301         }
00302     }
00303 }
00304 
00305 #endif
```

6.15 src/sigmath_test.cpp File Reference

```
#include <string>
#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"
```

Include dependency graph for sigmath_test.cpp:



Functions

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

6.15.1 Function Documentation

6.15.1.1 int main (int *argc*, char ** *argv*)

Definition at line 20 of file sigmath_test.cpp.

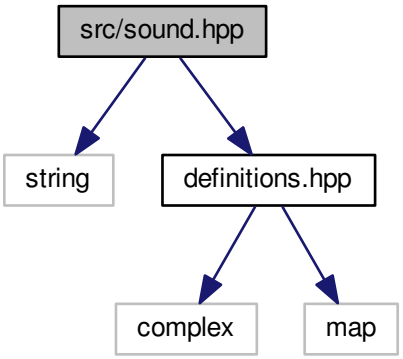
6.16 sigmath_test.cpp

```
00001
00007 #include <string>
00008
00009 #include "definitions.hpp"
00010 #include "fileio.hpp"
00011 #include "process.hpp"
00012
00013 using namespace std;
00014 using namespace vaso;
00015
00020 int main(int argc, char** argv) {
00021
00022 }
```

6.17 src/sound.hpp File Reference

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

Include dependency graph for sound.hpp:



Namespaces

- vaso

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

Functions

- void vaso::play (auto filename)

6.18 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, 8–10

decibels
    vaso, 10
diff
    vaso, 10

fft
    vaso, 11

index
    Maximum, 18

Left
    vaso, 8

mag
    vaso, 11
max
    vaso, 12
Maximum, 17
    index, 18
    value, 18

play
    vaso, 13
Process
    vaso, 13
process
    vaso, 14

Right
    vaso, 8

Side
    vaso, 8
smooth
    vaso, 16

value
    Maximum, 18
vaso, 7
    absolute, 8
    average, 8–10
    decibels, 10
    diff, 10
    fft, 11
    Left, 8
    mag, 11
    max, 12
    play, 13
    Process, 13
    process, 14
    Right, 8
    Side, 8
    smooth, 16