Andrew and Nick’s Project

Generated by Doxygen 1.8.8

Fri Apr 15 2016 02:50:06

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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*)

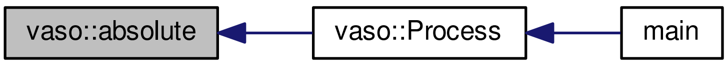
Ensures all elements in an array are positive. Note that this function replaces array elements if necessary. It does not populate a new array.

Parameters

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

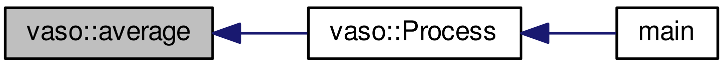
data	the array from which to compute the average
size	the number of elements in the data array

Returns

the computed average

Definition at line 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

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 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

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

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*)

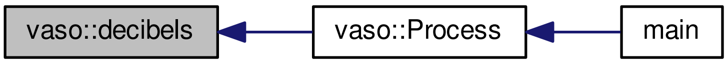
Converts an array of floats to "power decibels", i.e., x[n] = 20∗log10(x[n]). The decibel values are written to the same array that contained the values to be converted. In other words, this function should perform an in-place, element-wise conversion.

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 157 of file sigmath.hpp.

Here is the caller graph for this function:



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

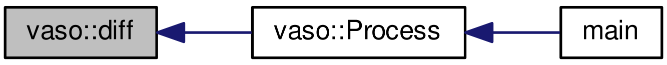
Computes the left-handed first derivative of a discrete signal. The first element will be 0.

Parameters

data	an array containing the discrete signal data
size	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.

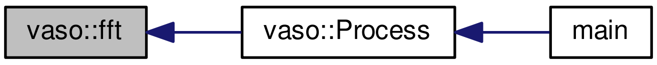
This code is based on code from `http://rosettacode.org/wiki/Fast_Fourier_transform#C.↩ 2B.2B`.

Parameters

data	the array whose values should be replaced with its DFT
size	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

dir	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*)

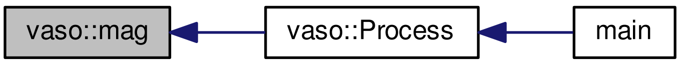
Computes the magitude of an array of complex numbers.

Parameters

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 newmag

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

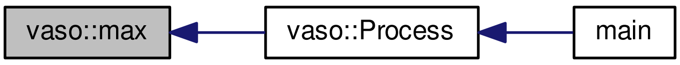
data	the array whose maximum value is to be found
uint32	size the number of elements in the data array

Returns

the maximum value and its index in a Maximum structure

Definition at line 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

filename	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 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.

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

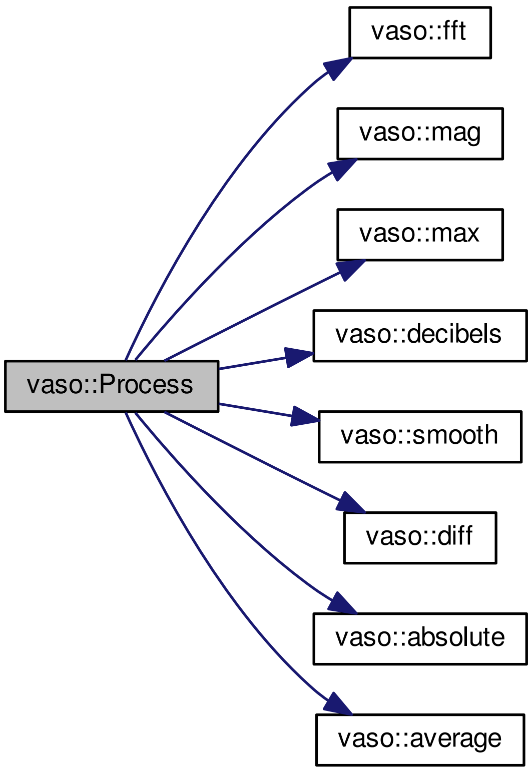
SAMPLE_CO↩ UNT	the number of samples in each recording. MUST be a power of two.
SAMPLE_FREQ	the sampling frequency in Hz or Samples/second

Returns

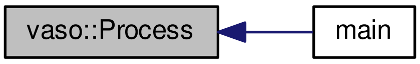
a map of the averaged left- and right-side parameters in DataParams structures

Definition at line 54 of file process.hpp.

Here is the call graph for this function:



Here is the caller graph for this function:



4.1.3.15 DataParams vaso::ReadParams (auto *filename*)

Reads the previously computed 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 64 of file fileio.hpp.

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

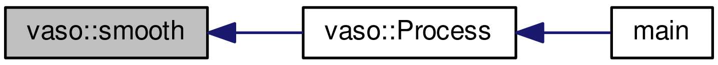
Applies an nth-order moving-average filter to a discrete signal.

Parameters

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 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

params	

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

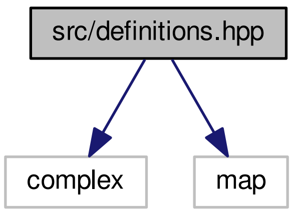
Chapter 6

File Documentation

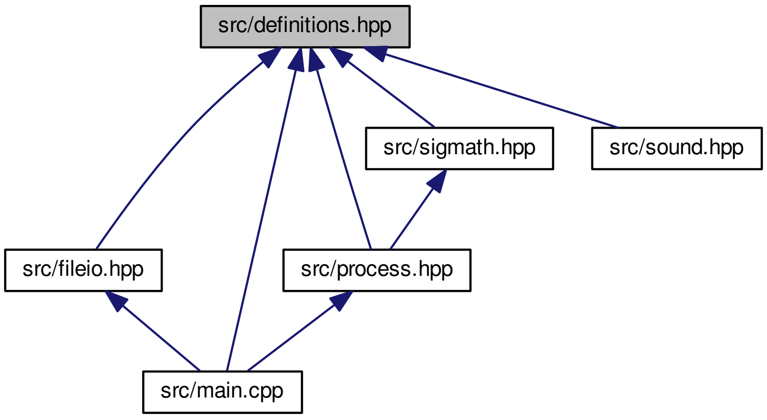
6.1 src/definitions.hpp File Reference

```
#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

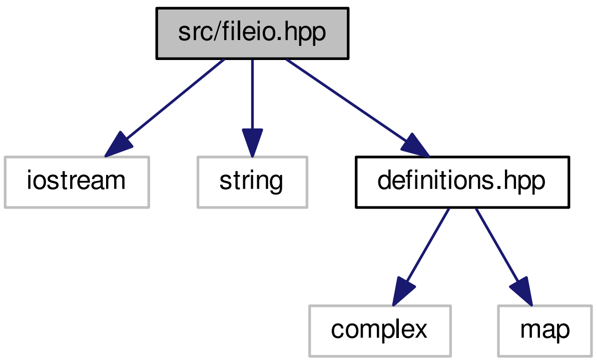
```
00001
00008 #ifndef definitions_H
00009 #define definitions_H
00010
00011 #include <complex>
00012 #include <map>
00013
00014 #define ERROR -1
00015 #define REC_COUNT 8
00016 #define SAMPLE_COUNT 262144
00017 #define SAMPLE_FREQ 48000
```

```
#define ENUM signed char

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;

typedef struct {
    float32 freq;
    float32 noise;
} DataParams;

typedef struct {
    float32 value;
    uint32 index;
} Maximum;

namespace vaso {
    enum class Side { Left, Right };
}

typedef struct {
    float32** data;
    uint8 recCount;
    uint32 sampleCount;
    uint32 sampleFreq;
    uint8* counter;
    std::map<vaso::Side, DataParams> results;
} ThreadParams;

#endif
```

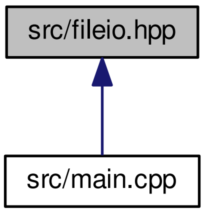
6.3 src/fileio.hpp File Reference

```
#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/"

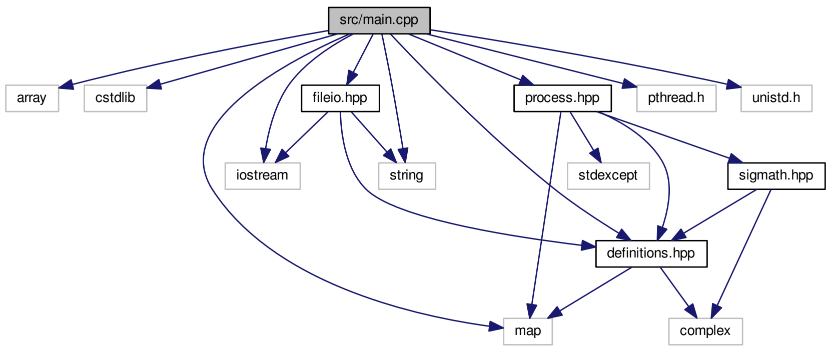
6.4 fileio.hpp

```
#ifndef fileio_H
#define fileio_H

#include <iostream>
#include <string>

#include "definitions.hpp"

namespace vaso {
    const std::string PATIENT_PATH = "/home/pi/patients/";

    std::string CurrentDataName() {

    }

    std::string InitialDataName(auto dir) {

    }

    std::string PatientName() {

    }

    DataParams ReadParams(auto filename) {

    }

    std::string WriteParams(DataParams params, auto filename) {

    }
}

#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)

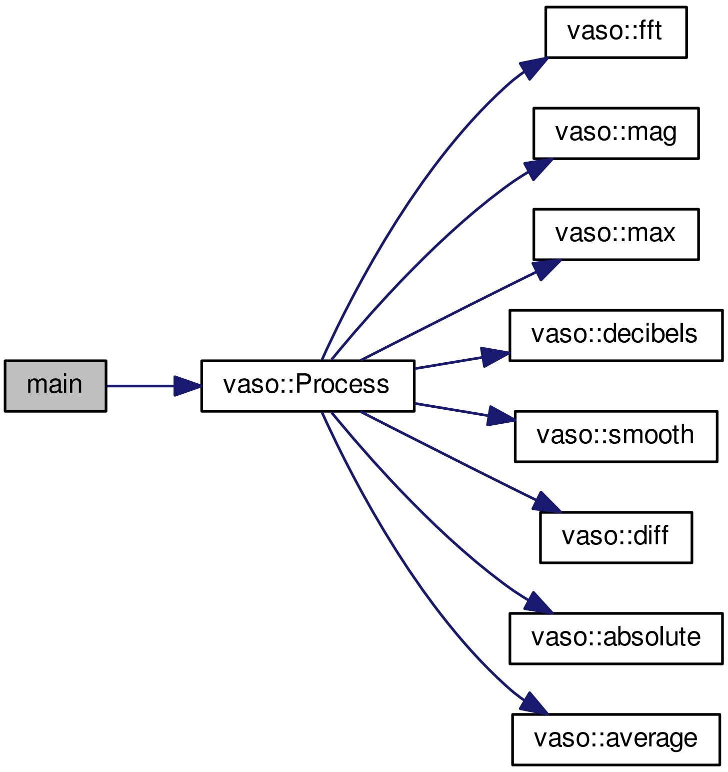
6.5.1 Function Documentation

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

The main program for this progject. 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:



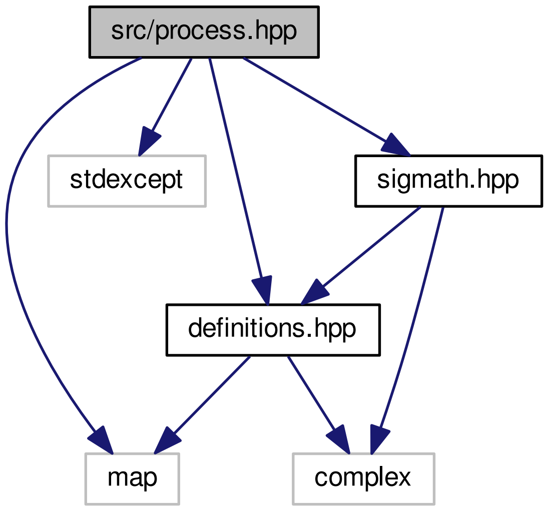
6.6 main.cpp

```
#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"

using namespace std;
using namespace vaso;

int main(int argc, char** argv) {
    // generate name for patient's file
    string filename = "";//PatientName();

    // TODO: Load all of patient's parameters

    // Record doppler audio
    float32 buffer[REC_COUNT][SAMPLE_COUNT];

    for(uint8 i = 0; i < REC_COUNT; i++) {
        // TODO: Prompt user to press ENTER to start recording

        int retSeek = 0;//fseek(STDIN_FILENO, 0, SEEK_END);
        int retRead = read(STDIN_FILENO, &buffer[i], SAMPLE_COUNT);

        if(retSeek != 0 || retRead < SAMPLE_COUNT) {
            cerr << "An error occurred reading the doppler audio! "
                "The program will now exit." << endl;
            return ERROR;
        }

        // TODO: Print message about recording stopped
    }

    map<Side, DataParams> results = Process(buffer);

    // TODO: Print results & probable diagnosis

    // TODO: Write all results to file
}
```

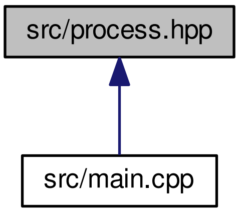
6.7 src/process.hpp File Reference

```
#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])

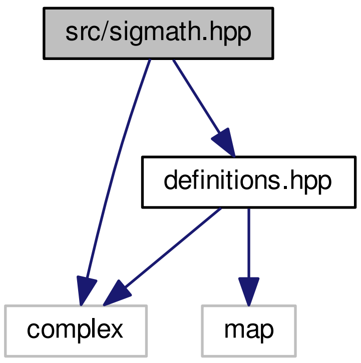
6.8 process.hpp

```
#ifndef process_H
#define process_H

#include <map>
#include <stdexcept>

#include "definitions.hpp"
#include "sigmath.hpp"

namespace vaso {
    std::map<Side, DataParams> Process(float32 data[REC_COUNT][
SAMPLE_COUNT]) {
        // just in case SAMPLE_COUNT isn't a power of two
        if((SAMPLE_COUNT & (SAMPLE_COUNT - 1) != 0) || SAMPLE_COUNT < 2) {
            throw std::invalid_argument(
                    "The number of samples is not a power of two!");
        }

        // declare function-scoped variables
        uint32 freqSize = SAMPLE_COUNT / 2;
        cfloat32 cdata[REC_COUNT][SAMPLE_COUNT];
        float32 fdata[REC_COUNT][freqSize];
        DataParams tempParams[REC_COUNT];
        std::map<Side, DataParams> sideParams;

        for(uint8 rCount = 0; rCount < REC_COUNT; rCount++) {
            // convert data to complex numbers for fft()
            for(uint32 i = 0; i < SAMPLE_COUNT; i++) {
                cdata[rCount][i] = data[rCount][i];
            }

            // find frequency spectrum in relative decibels
            fft(cdata[rCount], SAMPLE_COUNT);
            mag(cdata[rCount], fdata[rCount], freqSize);
            Maximum maximum = max(fdata[rCount], freqSize);

            for(uint32 i = 0; i < freqSize; i++) {
                fdata[rCount][i] /= maximum.value;
            }

            decibels(fdata[rCount], freqSize);

            /*
             * Run spectrum values through moving-average filter to smooth the
             * curve and make it easier to determine the derivative.
             */
            smooth(fdata[rCount], freqSize, 20);

            /*
             * Find the derivative of the smoothed spectrum. Bote that both this
             * filter and the previous are necessary to the algorithm.
             */
            diff(fdata[rCount], freqSize);
            smooth(fdata[rCount], freqSize, 100);
            absolute(fdata[rCount], freqSize);

            // find the parameters of this specific recording
            uint16 offset = 1000;
            absolute(&fdata[rCount][offset],    freqSize - offset);
            uint32 index = max(&fdata[rCount][offset],
                    freqSize - offset).index;
            tempParams[rCount].freq = index * (float)SAMPLE_FREQ / freqSize;
            tempParams[rCount].noise =
                average(&fdata[rCount][index + 2 * offset],
                        freqSize - 2 * offset);
        }

        // calculate the parameters for each side to be returned
        sideParams[Side::Left] = average(&tempParams[0], REC_COUNT / 2);
        sideParams[Side::Right] = average(&tempParams[REC_COUNT / 2],
                REC_COUNT / 2);
        return sideParams;
    }
}

#endif
```

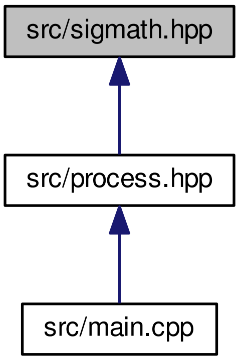
6.9 src/sigmath.hpp File Reference

```
#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

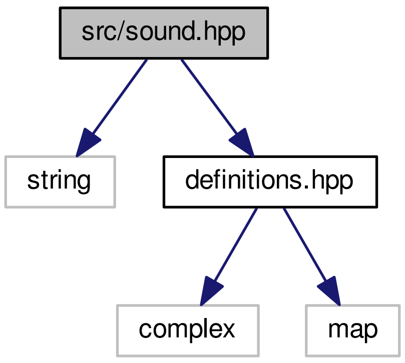
```

#ifndef sigmath_H
#define sigmath_H

#include <complex>

#include "definitions.hpp"

namespace vaso {
    // PROTOTYPES

    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);

    // DEFINITIONS

    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) {
        // data is an array. Access like so: data[index]
    }

    void decibels(float32* data, uint32 size) {
        for(uint32 i = 0; i < size; i++) {
            data[i] = 20 * log10(data[i]);
        }
    }

    void diff(float32* data, uint32 size) {

    }

    void fft(cfloat32* data, uint32 size) {
        // DFT
        uint32 k = size;
        uint32 n;
        float32 thetaT = M_PI / size;
        cfloat32 phiT(cos(thetaT), sin(thetaT));
        cfloat32 T;

        while(k > 1) {
            n = k;
            k >>= 1;
            phiT = phiT * phiT;
            T = 1.0L;

            for(uint32 l = 0; l < k; l++) {
                for(uint32 a = l; a < size; a += n) {
                    uint32 b = a + k;
                    cfloat32 t = data[a] -data[b];
                    data[a] +=data[b];
                    data[b] = t * T;
                }

                T *= phiT;
            }
        }

        // Decimate
        uint32 m = (uint32)log2(size);

        for(uint32 a = 0; a < size; a++) {
            uint32 b = a;

            // Reverse bits
            b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1));
            b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
            b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
            b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
            b = ((b >> 16) | (b << 16)) >> (32 - m);

            if (b > a)
            {
                cfloat32 t = data[a];
                data[a] =data[b];
                data[b] = t;
            }
        }
    }

    void mag(cfloat32* orig, float32* newmags, uint32 size) {

    }

    Maximum max(float32* data, uint32 size) {

    }

    void smooth(float32* data, uint32 size, uint16 order) {
        float32 coeff = 1 / (float32)order;
        float32 temp[size];

        for(uint32 i = 0; i < size; i++) {
            temp[i] = 0;

            for(uint16 j = 0; j < order && j <= i; j++) {
                temp[i] += data[i - j];
            }

            temp[i] *= coeff;
        }
    }
}

#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

```

#ifndef sound_H
#define sound_H

#include <string>

#include "definitions.hpp"

namespace vaso {
    void play(auto filename) {

    }
}

#endif
```

Index