# Andrew and Nick's Project

Generated by Doxygen 1.8.8

Tue Apr 19 2016 20:03:11

# Contents

# Chapter 1

# Bug List

**File fileio.hpp**

file is overly complicated and much more bug-prone

# Chapter 2

# Namespace Index

## 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

# Chapter 3

# Class Index

## 3.1 Class List

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

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

# Chapter 5

# Namespace Documentation

## 5.1 vaso Namespace Reference

### Enumerations

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

### Functions

- std::string PatientName ()
- std::map< Side, DataParams > ReadParams (auto filename)
- void WriteParams (std::map< Side, DataParams > myMap, auto filename)
- 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 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 CSV_HEADER = "Time,Side,Frequency,Noise Level"
- const std::string PATIENT_PATH = "/home/pi/patients/"

### 5.1.1 Detailed Description

This namespace contains all code related to this project.

### 5.1.2 Enumeration Type Documentation

#### 5.1.2.1 enum vaso::Side [strong]

Side of the head to which a recording pertains.

**Enumerator**

***Left***

***Right***

Definition at line 110 of file definitions.hpp.

```
00110 { Left, Right };
```

### 5.1.3 Function Documentation

#### 5.1.3.1 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.

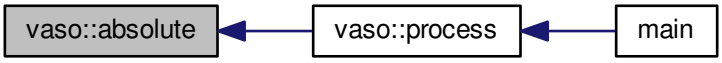
**Parameters**

| | |
|---|---|
| *data* | the array whose elements must all be positive |
| *size* | the number of elements in the data array |

Definition at line 123 of file sigmath.hpp.

```
00123                                                {
00124         for(uint32 i = 0; i < size; i++) {
00125             data[i] = fabsf(data[i]);
00126         }
00127     }
```

Here is the caller graph for this function:




#### 5.1.3.2 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**

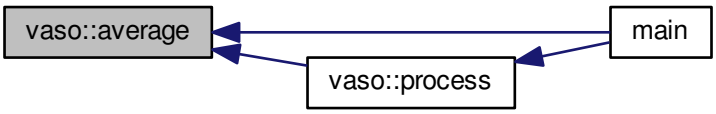
the computed average

Definition at line 129 of file sigmath.hpp.

```
00129                                                  {
00130         float32 ave;
00131
00132         for(uint32 i = 0; i < size; i++) {
```

```
00133             ave += data[i];
00134         }
00135
00136         ave = ave / size;
00137         return ave;
00138     }
```

Here is the caller graph for this function:




#### 5.1.3.3 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 140 of file sigmath.hpp.

```
00140                                                   {
00141         DataParams ave;
00142
00143         for(uint8 i = 0; i < size; i++) {
00144             //freq is an attribute. this is how to add structure attributes
00145             ave.freq += params[i].freq;
00146             ave.noise += params[i].noise;
00147         }
00148
00149         ave.freq /= size;
00150         ave.noise /= size;
00151         return ave;
00152     }
```

#### 5.1.3.4 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.

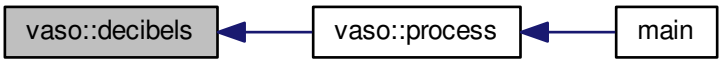
**Parameters**

| | |
|---|---|
| *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 154 of file sigmath.hpp.

```
00154                                                {
00155         for(uint32 i = 0; i < size; i++) {
00156             data[i] = 20 * log10(data[i]);
00157         }
00158     }
```

Here is the caller graph for this function:




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

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

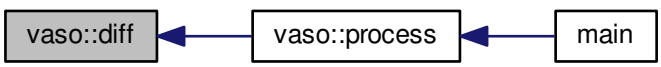
**Parameters**

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

Definition at line 160 of file sigmath.hpp.

```
00160                                           {
00161         float32 temp[size];
00162         temp[0] = 0;
00163
00164         for(uint32 i = 1; i < size; i++) {
00165             temp[i] = data[i] - data[i-1];
00166         }
00167
00168         for(uint32 i = 0; i < size; i++) {
00169             data[i] = temp[i];
00170         }
00171     }
```

Here is the caller graph for this function:



#### 5.1.3.6 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.

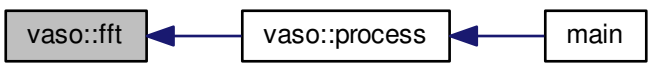
**Parameters**

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

Definition at line 173 of file sigmath.hpp.

```
00173                                           {
00174         // DFT
00175         uint32 k = size;
00176         uint32 n;
00177         float32 thetaT = M_PI / size;
00178         cfloat32 phiT(cos(thetaT), sin(thetaT));
00179         cfloat32 T;
00180 
00181         while(k > 1) {
00182             n = k;
00183             k >>= 1;
00184             phiT = phiT * phiT;
00185             T = 1.0L;
00186 
00187             for(uint32 l = 0; l < k; l++) {
00188                 for(uint32 a = l; a < size; a += n) {
00189                     uint32 b = a + k;
00190                     cfloat32 t = data[a] - data[b];
00191                     data[a] += data[b];
00192                     data[b] = t * T;
00193                 }
00194 
00195                 T *= phiT;
00196             }
00197         }
00198 
00199         // Decimate
00200         uint32 m = (uint32)log2(size);
00201 
00202         for(uint32 a = 0; a < size; a++) {
00203             uint32 b = a;
00204 
00205             // Reverse bits
00206             b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1));
00207             b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
00208             b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
00209             b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
00210             b = ((b >> 16) | (b << 16)) >> (32 - m);
00211 
00212             if (b > a)
00213             {
00214                 cfloat32 t = data[a];
00215                 data[a] = data[b];
00216                 data[b] = t;
00217             }
00218         }
00219     }
```

Here is the caller graph for this function:



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

Computes the magitude of an array of complex numbers.

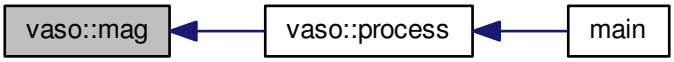
**Parameters**

| | |
|---|---|
| *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 221 of file sigmath.hpp.

```
00221                                                       {
00222         //loop to run throught the length of array orig
00223         for(uint32 n = 0; n < size; n++) {
00224             /*
00225              * abs should calculate the magnitude of complex array elements.
00226              * saves to new array
00227              */
00228             newmags[n] = std::abs(orig[n]);
00229         }
00230     }
```

Here is the caller graph for this function:




#### 5.1.3.8 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 |

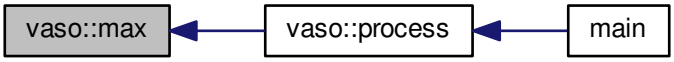
**Returns**

the maximum value and its index in a Maximum structure

Definition at line 232 of file sigmath.hpp.

```
00232                                           {
00233         Maximum m;
00234
00235         //loop to run through the length of array data
00236         for (uint32 i = 0; i < size; i++) {
00237             /*
00238              * when value at data[i] is above max.value,
00239              * sets max.value equal to data[i] and max.index equal to i
00240              */
00241             if (data[i] > m.value) {
00242                 m.value = data[i];
00243                 m.index = i;
00244             }
00245         }
00246
00247         return m;
00248     }
```

Here is the caller graph for this function:




#### 5.1.3.9 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. A newly created file will contain the CSV header for the file's data.

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

**Returns**

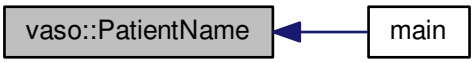
the file under which all patient data is saved

Definition at line 43 of file fileio.hpp.

```
00043                           {
00044         std::string fname = "";
00045         std::string mname = "";
00046         std::string lname = "";
00047         std::string patfil = "";
00048         std::string patientname = "";
00049         uint32 track1 = 0;
00050         uint32 track2 = 0;
00051         uint32 track3 = 0;
00052
00053         do {
00054             std::cout << "Please enter the patients name." << std::endl;
00055             std::cout << "First name: ";
00056             std::cin >> fname;
00057             std::cout << "Middle name: ";
00058             std::cin >> mname;
00059             std::cout << "Last name: ";
00060             std::cin >> lname;
00061
00062             // creates new std::string with path to patient file
00063             patientname = PATIENT_PATH + lname + ", " + fname
00064                 + " " + mname + ".csv";
00065
00066             // prints out patientname. shows user the path to the patient file
00067             std::cout << patientname << std::endl << std::endl;
00068             std::ifstream file(patientname.c_str());
00069
00070             if (file.good()) {
00071                 track1 = 1;
00072             }
00073
00074             /*
00075              * Compares patientname to existing files and lets user know
00076              * if the file does not exist.
00077              */
00078             else if (!file.good()) {
00079                 /*
00080                  * Do while statement to continue asking user about the file
00081                  * if their input is not acceptable
00082                  */
00083                 do {
00084                     std::cout << "Patient file does not exist, would you like "
00085                         "to create file or re-enter their name?" << std::endl;
00086                     std::cout << "  *Type 'create' and press enter key "
00087                         "to create the patient file." << std::endl;
00088                     std::cout << "  *Type 'reenter' and press enter key "
00089                         "to re-enter the patients name." << std::endl;
00090                     std::cout << std::endl;
```

```
00091                     std::cin >> patfil;
00092
00093                     /*
00094                      * patfil equals create, track1 and 2 will increase
00095                      * escaping both do while loops
00096                      */
00097                     if(patfil == "create") {
00098                         std::ofstream createfile(patientname.c_str());
00099                         track1 = 1;
00100                        track2 = 1;
00101                        track3 = 1;
00102                        createfile << CSV_HEADER << std::endl;
00103                        createfile.flush();
00104                        createfile.close();
00105                    }
00106
00107                    /*
00108                     *patfil equals renter, track1 will remain zero allowing
00109                     *user to reenter the patient name.
00110                     */
00111                    else if(patfil == "reenter") {
00112                        track1 = 0;
00113                        track2 = 1;
00114                    }
00115
00116                    /*
00117                     *The users input was neither create or reenter. User
00118                     *must enter patient name again.
00119                     */
00120                    else {
00121                        std::cout << std::endl;
00122                        std::cout << "Your input is not acceptable." << std::endl;
00123                        std::cout << std::endl;
00124                    }
00125                }while(track2 == 0);
00126            }
00127        } while (track1 == 0);
00128
00129        return patientname; //returns the path to the patient file
00130    }
```

Here is the caller graph for this function:




#### 5.1.3.10 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 20 of file sound.hpp.

```
00020                            {
00021
00022    }
```

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

Analyzes a single recording to determine the drop-off frequency and average noiseband noise power.

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* | array containing float32 samples of audio |
| *size* | number of samples in each recording. MUST be a power of two. |
| *samplingRate* | the sampling frequency in Hz or Samples/second |

**Returns**

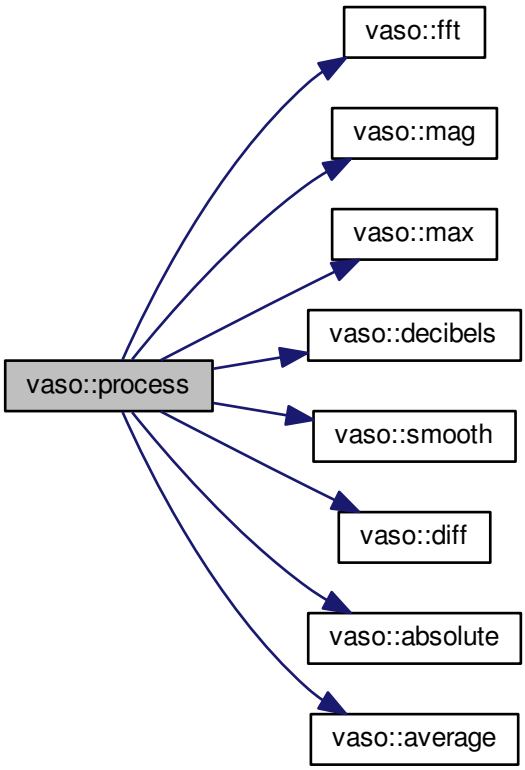
cut-off frequency (Hz) and average noiseband noise power in decibels

Definition at line 48 of file process.hpp.

```
00048                                                                                {
00049         if((size & (size - 1) != 0) || size < 2) {
00050             throw std::invalid_argument(
00051                     "The number of samples is not a power of two!");
00052         }
00053 
00054         // declare function-scoped variables
00055         uint32 freqSize = size / 2;
00056         cfloat32* cdata = (cfloat32*)std::malloc(size * sizeof(
      cfloat32));
00057         float32* fdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00058         float32* origdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00059 
00060         // convert data to complex numbers for fft()
00061         for(uint32 i = 0; i < size; i++) {
00062             cdata[i] = data[i];
00063         }
00064 
00065         // find frequency spectrum in relative decibels
00066         fft(cdata, size);
00067         mag(cdata, fdata, freqSize);
00068         Maximum maximum = max(fdata, freqSize);
00069 
00070         for(uint32 i = 0; i < freqSize; i++) {
00071             fdata[i] /= maximum.value;
00072         }
00073 
00074         decibels(fdata, freqSize);
00075 
00076         for(uint32 i = 0; i < freqSize; i++) {
00077             origdata[i] = fdata[i];
00078         }
00079 
00080         /*
00081          * Run spectrum values through moving-average filter to smooth the
00082          * curve and make it easier to determine the derivative.
00083          */
00084         smooth(fdata, freqSize, 20);
00085 
00086         /*
00087          * Find the derivative of the smoothed spectrum. Bote that both this
00088          * filter and the previous are necessary to the algorithm.
00089          */
00090         diff(fdata, freqSize);
00091         smooth(fdata, freqSize, 100);
00092         absolute(fdata, freqSize);
00093 
00094         // find the parameters of this specific recording
00095         uint16 offset = 1000;
00096         absolute(&fdata[offset], freqSize - offset);
00097         maximum = max(&fdata[offset], freqSize - offset);
00098         uint32 index = maximum.index + offset;
```

```
00099
00100         DataParams params;
00101         params.freq = index * (float)SAMPLE_FREQ / freqSize / 2;
00102         params.noise = average(&origdata[index + offset],
00103                 freqSize - offset - index);
00104
00105         free(cdata);
00106         free(fdata);
00107
00108         return params;
00109
00110     }
```

Here is the call graph for this function:




Here is the caller graph for this function:

#### 5.1.3.12 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

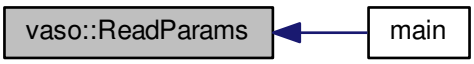
the patient parameters read for each side

Definition at line 141 of file fileio.hpp.

```
00141                                                       {
00142         std::map<Side, DataParams> myMap;
00143         DataParams leftparams;
00144         DataParams rightparams;
00145 
00146         std::ifstream file(filename.c_str());
00147         std::string leftline;
00148         std::string rightline;
00149         std::string leftsearch = "Left";
00150         std::string rightsearch = "Right";
00151         std::string paramstring;
00152         std::string lfreqstr;
00153         std::string lnoisestr;
00154         std::string rfreqstr;
00155         std::string rnoisestr;
00156         uint32 lcnt = 0;
00157         uint32 rcnt = 0;
00158         float32 lfreqval;
00159         float32 lnoiseval;
00160         float32 rfreqval;
00161         float32 rnoiseval;
00162 
00163         /*
00164          * if statement which uses ifstream function to open patient file
00165          * filename)
00166          */
00167         if(file.is_open()) {
00168             /*
00169              * While statement to find the first Left line and save to
00170              *leftline as string.
00171              */
00172             while (getline(file, leftline)) {
00173                 if(leftline.find(leftsearch, 0) != std::string::npos) {
00174                     break;
00175                 }
00176 
00177             }
00178 
00179             /*
00180              * While statement to find first right line and save to rightline
00181              * as string.
00182              */
00183             while (getline(file,rightline)) {
00184                 if(rightline.find(rightsearch, 0) != std::string::npos) {
00185                     break;
00186                 }
00187             }
00188 
00189             // Code to break leftline and rightline into its parts
00190             std::stringstream lss(leftline);
00191             std::stringstream rss(rightline);
00192 
00193             while(getline(lss,paramstring, ',')) {
00194                 lcnt++;
00195 
00196                 if(lcnt == 3) {
00197                     lfreqstr = paramstring;
00198                 }
00199 
00200                 else if(lcnt == 4) {
00201                     lnoisestr = paramstring;
00202                 }
00203             }
00204 
00205             while(getline(rss,paramstring, ',')) {
00206                 rcnt++;
00207 
00208                 if(rcnt == 3) {
00209                     rfreqstr = paramstring;
00210                 }
00211 
00212                 else if(rcnt == 4) {
00213                     rnoisestr = paramstring;
```

```
00214                 }
00215             }
00216
00217             /*
00218              * Statement to convert lfreq, lnoise, rfreq, and rnoise from
00219              * strings to floats.
00220              * */
00221             lfreqval = atof(lfreqstr.c_str());
00222             lnoiseval = atof(lnoisestr.c_str());
00223             rfreqval = atof(rfreqstr.c_str());
00224             rnoiseval = atof(rnoisestr.c_str());
00225
00226             file.close();
00227         }
00228
00229         else {
00230             throw std::runtime_error("The patient file could not be opened.");
00231         }
00232
00233         leftparams.freq = lfreqval;
00234         leftparams.noise = lnoiseval;
00235         rightparams.freq = rfreqval;
00236         rightparams.noise = rnoiseval;
00237
00238         myMap[Side::Left] = leftparams;
00239         myMap[Side::Right] = rightparams;
00240
00241         return myMap;
00242     }
```

Here is the caller graph for this function:




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

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

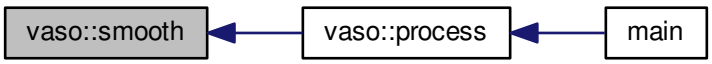
**Parameters**

| | |
|---|---|
| *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 250 of file sigmath.hpp.

```
00250                                                       {
00251         float32 coeff = 1 / (float32)order;
00252         float32 temp[size];
00253
00254         for(uint32 i = 0; i < size; i++) {
00255             temp[i] = 0;
00256
00257             for(uint16 j = 0; j < order && j <= i; j++) {
00258                 temp[i] += data[i - j];
00259             }
00260
00261             temp[i] *= coeff;
00262         }
00263
00264         for(uint32 i = 0; i < size; i++) {
00265             data[i] = temp[i];
00266         }
00267     }
```

Here is the caller graph for this function:




#### 5.1.3.14 void vaso::WriteParams ( std::map< Side, DataParams > *myMap*, auto *filename* )

Writes (appends) the passed parameters to the specified file.

**Parameters**

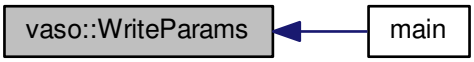
| | |
|---|---|
| *myMap* | contains the parameters to be written |

the patient CSV file's filename

Definition at line 251 of file fileio.hpp.

```
00251                                                                    {
00252         char temp[80];
00253         std::ofstream file(filename.c_str(),
00254                 std::ofstream::out | std::ofstream::app);
00255
00256         //Gives pointer measurementtime a data type of time_t.
00257         time_t measurementtime;
00258         time(&measurementtime); //Gets the current time.
00259         strftime(temp, 80, "%c", localtime(&measurementtime));
00260         std::string fTime = std::string(temp);
00261
00262         //if statement to print the Left side parameters to the patient file.
00263         if(file.is_open()) {
00264             file << fTime + "," + "Left" + ","
00265                 + std::to_string(myMap[Side::Left].freq)
00266                 + ", " + std::to_string(myMap[Side::Left].noise) << std::endl;
00267         }
00268
00269         //if statement to print the Right side parameters to the patient file.
00270         if(file.is_open()) {
00271             file << fTime + "," + "Right" + ","
00272                 + std::to_string(myMap[Side::Right].freq)
00273                 + ", " + std::to_string(myMap[Side::Right].noise) << std::endl;
00274         }
00275
00276         else {
00277             std::cout << "Patient file can not be opened!" << std::endl;
00278         }
00279
00280         file.close();
00281     }
```

Here is the caller graph for this function:



### 5.1.4 Variable Documentation

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

First line of CSV data file, which declares columns.

Definition at line 25 of file fileio.hpp.

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

Absolute path to the folder containing the patients' data

Definition at line 30 of file fileio.hpp.

# Chapter 6

# Class Documentation

## 6.1 DataParams Struct Reference

`#include <definitions.hpp>`

### Public Attributes

- float32 freq = 0
- float32 noise = 0

### 6.1.1 Detailed Description

Calculated results from processing the audio recordings.

Definition at line 86 of file definitions.hpp.

### 6.1.2 Member Data Documentation

#### 6.1.2.1 float32 DataParams::freq = 0

Definition at line 87 of file definitions.hpp.

#### 6.1.2.2 float32 DataParams::noise = 0

Definition at line 88 of file definitions.hpp.

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

- src/definitions.hpp

## 6.2 Maximum Struct Reference

`#include <definitions.hpp>`

### Public Attributes

- float32 value = 0
- uint32 index = 0

### 6.2.1 Detailed Description

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

Definition at line 95 of file definitions.hpp.

### 6.2.2 Member Data Documentation

#### 6.2.2.1 uint32 Maximum::index = 0

Definition at line 97 of file definitions.hpp.

#### 6.2.2.2 float32 Maximum::value = 0

Definition at line 96 of file definitions.hpp.

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

- src/definitions.hpp

# Chapter 7

# File Documentation

## 7.1 etc/doxygen.config File Reference

Contains Doxygen configuration settings.

### 7.1.1 Detailed Description

Contains Doxygen configuration settings.

**Author**

Samnuel Andrew Wisner, awisner94@gmail.com

Definition in file doxygen.config.

## 7.2 doxygen.config

```
00001 PROJECT_NAME = "Andrew and Nick's Project"
00002
00003 INPUT = src/ etc/doxygen.config makefile
00004 OUTPUT_DIRECTORY = doc/
00005
00006 GENERATE_HTML = YES
00007 GENERATE_RTF = YES
00008 GENERATE_LATEX = YES
00009 GENERATE_MAN = YES
00010 GENERATE_XML = NO
00011 GENERATE_DOCBOOK = NO
00012
00013 USE_PDF_LATEX = YES
00014 USE_PDF_HYPERLINKS = YES
00015
00016 RECURSIVE = YES
00017 SOURCE_BROWSER = YES
00018 SOURCE_TOOLTIPS = YES
00019 EXTRACT_ALL = YES
00020 DISABLE_INDEX = NO
00021 GENERATE_TREEVIEW = YES
00022 SEARCHENGINE = YES
00023 SERVER_BASED_SEARCH = NO
00024
00025 LATEX_SOURCE_CODE = YES
00026 STRIP_CODE_COMMENTS = YES
00027 INLINE_SOURCES = YES
00028
00029 HAVE_DOT = YES
00030 CALL_GRAPH = YES
00031 CALLER_GRAPH = YES
```

## 7.3 makefile File Reference

Contains recipes for building the test applications, the main application, and the documentation.

### 7.3.1 Detailed Description

Contains recipes for building the test applications, the main application, and the documentation.

**Author**

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file makefile.

## 7.4 makefile

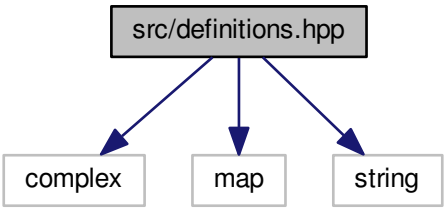
```
00001 GCC = g++ -g -std=gnu++14
00002
00003 count:
00004    grep -r "src/" -e "Samuel Andrew Wisner" -l | xargs wc -l
00005
00006 docs:
00007    rm -r doc/
00008    doxygen etc/doxygen.config
00009    cd doc/latex; make pdf;
00010    git reset
00011    git add doc/.
00012    git commit -m "Updated documentation."
00013    git push
00014
00015 fileio-test:
00016    $(GCC) src/fileio_test.cpp -o bin/fileiotest
00017
00018 patient-name-test:
00019    $(GCC) src/patient_name_test.cpp -o bin/patnametest
00020
00021 process-test:
00022    $(GCC) src/process_test.cpp -o bin/proctest
00023
00024 read-params-test:
00025    $(GCC) src/read_params_test.cpp -o bin/rptest
00026
00027 stdin-clear-test:
00028    $(GCC) src/stdin_clear_test.cpp -o bin/cleartest
00029
00030 vaso:
00031    $(GCC) src/main.cpp -o bin/vaso
00032
```

## 7.5 src/definitions.hpp File Reference

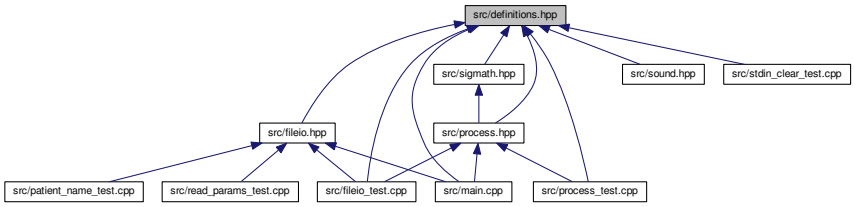
Contains declarations of system-independant (universal size) integers and float types, shortened type names for some commonly used types, and enumerations.

```
#include <complex>
#include <map>
#include <string>
```

Include dependency graph for definitions.hpp:




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




## Classes

- struct DataParams
- struct Maximum

## Namespaces

- vaso

## Macros

- #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 }

### Variables

- const uint8 DURATION = 6
- const sint8 ERROR = -1
- const uint8 REC_COUNT = 6
- const uint32 SAMPLE_COUNT = 262144
- const uint16 SAMPLE_FREQ = 48000
- const std::string TEMP_FILE = ".temp"
- const uint32 BUFFER_SIZE = SAMPLE_COUNT * sizeof(float32)

## 7.5.1 Detailed Description

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 in file definitions.hpp.

## 7.5.2 Macro Definition Documentation

### 7.5.2.1 #define ENUM signed char

Definition at line 16 of file definitions.hpp.

## 7.5.3 Typedef Documentation

### 7.5.3.1 typedef unsigned char byte

Definition at line 20 of file definitions.hpp.

### 7.5.3.2 typedef std::complex<float32> cfloat32

Complex float32's.

Definition at line 81 of file definitions.hpp.

### 7.5.3.3 typedef float float32

Definition at line 33 of file definitions.hpp.

#### 7.5.3.4 typedef double float64

Definition at line 34 of file definitions.hpp.

#### 7.5.3.5 typedef signed short sint16

Definition at line 25 of file definitions.hpp.

#### 7.5.3.6 typedef signed int sint32

Definition at line 28 of file definitions.hpp.

#### 7.5.3.7 typedef signed long long sint64

Definition at line 31 of file definitions.hpp.

#### 7.5.3.8 typedef signed char sint8

Definition at line 22 of file definitions.hpp.

#### 7.5.3.9 typedef unsigned short uint16

Definition at line 24 of file definitions.hpp.

#### 7.5.3.10 typedef unsigned int uint32

Definition at line 27 of file definitions.hpp.

#### 7.5.3.11 typedef unsigned long long uint64

Definition at line 30 of file definitions.hpp.

#### 7.5.3.12 typedef unsigned char uint8

Definition at line 21 of file definitions.hpp.

### 7.5.4 Variable Documentation

#### 7.5.4.1 const uint32 BUFFER_SIZE = SAMPLE_COUNT ∗ sizeof(float32)

Size of the sample buffer.

Definition at line 73 of file definitions.hpp.

#### 7.5.4.2 const uint8 DURATION = 6

Duration of recording in seconds.

Definition at line 42 of file definitions.hpp.

#### 7.5.4.3 const sint8 ERROR = -1

Error integer returned when the program must exit with an error.

Definition at line 47 of file definitions.hpp.

#### 7.5.4.4 const uint8 REC_COUNT = 6

Number of recordings (both left and right) to make.

Definition at line 52 of file definitions.hpp.

#### 7.5.4.5 const uint32 SAMPLE_COUNT = 262144

Number of samples to use in processing the recordings. Must be a power of two. SAMPLE_COUNT / SAMPLE_↩ FREQ $<$ DURATION must be true.

Definition at line 58 of file definitions.hpp.

#### 7.5.4.6 const uint16 SAMPLE_FREQ = 48000

Recording sampling rate in Hz (NOT kHz).

Definition at line 63 of file definitions.hpp.

#### 7.5.4.7 const std::string TEMP_FILE = ".temp"

Filename of the temporary recording file.

Definition at line 68 of file definitions.hpp.

## 7.6 definitions.hpp

```
00001
00009 #ifndef definitions_H
00010 #define definitions_H
00011
00012 #include <complex>
00013 #include <map>
00014 #include <string>
00015
00016 #define ENUM signed char
00017
00018 // Type definitions
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
00036
00037 // Constants
00038
00042 const uint8 DURATION = 6;
00043
00047 const sint8 ERROR = -1;
00048
```

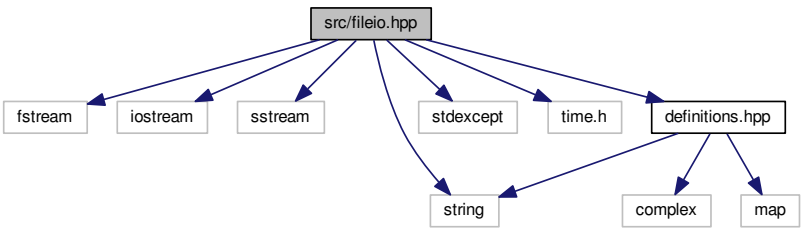
```
00052 const uint8 REC_COUNT = 6;
00053
00058 const uint32 SAMPLE_COUNT = 262144;
00059
00063 const uint16 SAMPLE_FREQ = 48000;
00064
00068 const std::string TEMP_FILE = ".temp";
00069
00073 const uint32 BUFFER_SIZE = SAMPLE_COUNT * sizeof(
      float32);
00074
00075
00076 // Objective/structural type definitions
00077
00081 typedef std::complex<float32> cfloat32;
00082
00086 typedef struct {
00087     float32 freq = 0;
00088     float32 noise = 0;
00089 } DataParams;
00090
00095 typedef struct {
00096     float32 value = 0;
00097     uint32 index = 0;
00098 } Maximum;
00099
00100
00101 // Enumerations
00102
00106 namespace vaso {
00110     enum class Side { Left, Right };
00111 }
00112
00113
00114 // Doxygen documentation for other files.
00115
00129 #endif
```

## 7.7 src/fileio.hpp File Reference

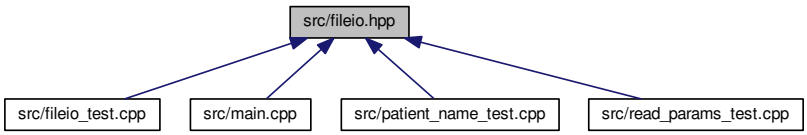
contains functions related to the file I/O use in this program

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <stdexcept>
#include <time.h>
#include "definitions.hpp"
```

Include dependency graph for fileio.hpp:



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




### Namespaces

- vaso

### Functions

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

### Variables

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

#### 7.7.1 Detailed Description

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

**Author**

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

**Bug** file is overly complicated and much more bug-prone

Definition in file fileio.hpp.

## 7.8 fileio.hpp

```
00001
00009 #ifndef fileio_H
00010 #define fileio_H
00011
00012 #include <fstream>
00013 #include <iostream>
00014 #include <sstream>
00015 #include <string>
00016 #include <stdexcept>
00017 #include <time.h>
00018
00019 #include "definitions.hpp"
00020
00021 namespace vaso {
00025     const std::string CSV_HEADER = "Time,Side,Frequency,Noise Level";
00026
```

```
00030     const std::string PATIENT_PATH = "/home/pi/patients/";
00031
00043     std::string PatientName() {
00044         std::string fname = "";
00045         std::string mname = "";
00046         std::string lname = "";
00047         std::string patfil = "";
00048         std::string patientname = "";
00049         uint32 track1 = 0;
00050         uint32 track2 = 0;
00051         uint32 track3 = 0;
00052
00053         do {
00054             std::cout << "Please enter the patients name." << std::endl;
00055             std::cout << "First name: ";
00056             std::cin >> fname;
00057             std::cout << "Middle name: ";
00058             std::cin >> mname;
00059             std::cout << "Last name: ";
00060             std::cin >> lname;
00061
00062             // creates new std::string with path to patient file
00063             patientname = PATIENT_PATH + lname + ", " + fname
00064                 + " " + mname + ".csv";
00065
00066             // prints out patientname. shows user the path to the patient file
00067             std::cout << patientname << std::endl << std::endl;
00068             std::ifstream file(patientname.c_str());
00069
00070             if (file.good()) {
00071                 track1 = 1;
00072             }
00073
00074             /*
00075              * Compares patientname to existing files and lets user know
00076              * if the file does not exist.
00077              */
00078             else if (!file.good()) {
00079                 /*
00080                  * Do while statement to continue asking user about the file
00081                  * if their input is not acceptable
00082                  */
00083                 do {
00084                     std::cout << "Patient file does not exist, would you like "
00085                         "to create file or re-enter their name?" << std::endl;
00086                     std::cout << "  *Type 'create' and press enter key "
00087                         "to create the patient file." << std::endl;
00088                     std::cout << "  *Type 'reenter' and press enter key "
00089                         "to re-enter the patients name." << std::endl;
00090                     std::cout << std::endl;
00091                     std::cin >> patfil;
00092
00093                     /*
00094                      * patfil equals create, track1 and 2 will increase
00095                      * escaping both do while loops
00096                      */
00097                     if(patfil == "create") {
00098                         std::ofstream createfile(patientname.c_str());
00099                         track1 = 1;
00100                         track2 = 1;
00101                         track3 = 1;
00102                         createfile << CSV_HEADER << std::endl;
00103                         createfile.flush();
00104                         createfile.close();
00105                     }
00106
00107                     /*
00108                      *patfil equals renter, track1 will remain zero allowing
00109                      *user to reenter the patient name.
00110                      */
00111                     else if(patfil == "reenter") {
00112                         track1 = 0;
00113                         track2 = 1;
00114                     }
00115
00116                     /*
00117                      *The users input was neither create or reenter. User
00118                      *must enter patient name again.
00119                      */
00120                     else {
00121                         std::cout << std::endl;
00122                         std::cout << "Your input is not acceptable." << std::endl;
00123                         std::cout << std::endl;
00124                     }
00125                 }while(track2 == 0);
00126             }
00127         } while (track1 == 0);
```

```
00128
00129         return patientname; //returns the path to the patient file
00130     }
00131
00141     std::map<Side, DataParams> ReadParams(auto filename) {
00142         std::map<Side, DataParams> myMap;
00143         DataParams leftparams;
00144         DataParams rightparams;
00145
00146         std::ifstream file(filename.c_str());
00147         std::string leftline;
00148         std::string rightline;
00149         std::string leftsearch = "Left";
00150         std::string rightsearch = "Right";
00151         std::string paramstring;
00152         std::string lfreqstr;
00153         std::string lnoisestr;
00154         std::string rfreqstr;
00155         std::string rnoisestr;
00156         uint32 lcnt = 0;
00157         uint32 rcnt = 0;
00158         float32 lfreqval;
00159         float32 lnoiseval;
00160         float32 rfreqval;
00161         float32 rnoiseval;
00162
00163         /*
00164          * if statement which uses ifstream function to open patient file
00165          * filename)
00166          */
00167         if(file.is_open()) {
00168             /*
00169              * While statement to find the first Left line and save to
00170              *leftline as string.
00171              */
00172             while (getline(file, leftline)) {
00173                 if(leftline.find(leftsearch, 0) != std::string::npos) {
00174                     break;
00175                 }
00176
00177             }
00178
00179             /*
00180              * While statement to find first right line and save to rightline
00181              * as string.
00182              */
00183             while (getline(file,rightline)) {
00184                 if(rightline.find(rightsearch, 0) != std::string::npos) {
00185                     break;
00186                 }
00187             }
00188
00189             // Code to break leftline and rightline into its parts
00190             std::stringstream lss(leftline);
00191             std::stringstream rss(rightline);
00192
00193             while(getline(lss,paramstring, ',')) {
00194                 lcnt++;
00195
00196                 if(lcnt == 3) {
00197                     lfreqstr = paramstring;
00198                 }
00199
00200                 else if(lcnt == 4) {
00201                     lnoisestr = paramstring;
00202                 }
00203             }
00204
00205             while(getline(rss,paramstring, ',')) {
00206                 rcnt++;
00207
00208                 if(rcnt == 3) {
00209                     rfreqstr = paramstring;
00210                 }
00211
00212                 else if(rcnt == 4) {
00213                     rnoisestr = paramstring;
00214                 }
00215             }
00216
00217             /*
00218              * Statement to convert lfreq, lnoise, rfreq, and rnoise from
00219              * strings to floats.
00220              * */
00221             lfreqval = atof(lfreqstr.c_str());
00222             lnoiseval = atof(lnoisestr.c_str());
00223             rfreqval = atof(rfreqstr.c_str());
```

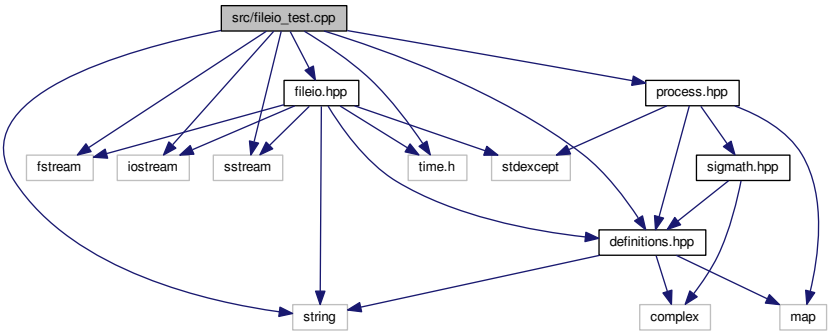
```
00224             rnoiseval = atof(rnoisestr.c_str());
00225
00226             file.close();
00227         }
00228
00229         else {
00230             throw std::runtime_error("The patient file could not be opened.");
00231         }
00232
00233         leftparams.freq = lfreqval;
00234         leftparams.noise = lnoiseval;
00235         rightparams.freq = rfreqval;
00236         rightparams.noise = rnoiseval;
00237
00238         myMap[Side::Left] = leftparams;
00239         myMap[Side::Right] = rightparams;
00240
00241         return myMap;
00242     }
00243
00251     void WriteParams(std::map<Side, DataParams> myMap, auto filename) {
00252         char temp[80];
00253         std::ofstream file(filename.c_str(),
00254                 std::ofstream::out | std::ofstream::app);
00255
00256         //Gives pointer measurementtime a data type of time_t.
00257         time_t measurementtime;
00258         time(&measurementtime); //Gets the current time.
00259         strftime(temp, 80, "%c", localtime(&measurementtime));
00260         std::string fTime = std::string(temp);
00261
00262         //if statement to print the Left side parameters to the patient file.
00263         if(file.is_open()) {
00264             file << fTime + "," + "Left" + ","
00265                 + std::to_string(myMap[Side::Left].freq)
00266                 + ", " + std::to_string(myMap[Side::Left].noise) << std::endl;
00267         }
00268
00269         //if statement to print the Right side parameters to the patient file.
00270         if(file.is_open()) {
00271             file << fTime + "," + "Right" + ","
00272                 + std::to_string(myMap[Side::Right].freq)
00273                 + ", " + std::to_string(myMap[Side::Right].noise) << std::endl;
00274         }
00275
00276         else {
00277             std::cout << "Patient file can not be opened!" << std::endl;
00278         }
00279
00280         file.close();
00281     }
00282 }
00283
00284 #endif
```

## 7.9 src/fileio_test.cpp File Reference

Contains program that tests the functions in fileio.hpp.

```
#include <fstream>
#include <iostream>
#include <sstream>
#include <string>
#include <time.h>
#include "definitions.hpp"
#include "fileio.hpp"
#include "process.hpp"
```

Include dependency graph for fileio_test.cpp:




## Functions

- int main ()

### 7.9.1 Detailed Description

Contains program that tests the functions in fileio.hpp.

#### Author

Samuel Andrew Wisner
Nicholas K. Nolan

Definition in file fileio_test.cpp.

### 7.9.2 Function Documentation

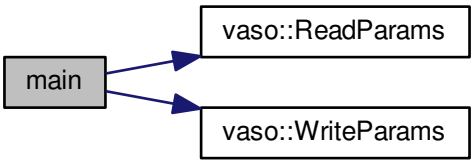
#### 7.9.2.1 int main ( )

Tests the functions in fileio.hpp.

Definition at line 24 of file fileio_test.cpp.

```
00024           {
00025     string path = PATIENT_PATH + "wizmack, sammy andy.csv";
00026     map<Side, DataParams> laMap = ReadParams(path);
00027     cout <<  laMap[Side::Right].freq << endl;
00028     cout << laMap[Side::Right].noise << endl;
00029
00030     WriteParams(laMap, path);
00031 }
```

Here is the call graph for this function:




## 7.10 fileio_test.cpp

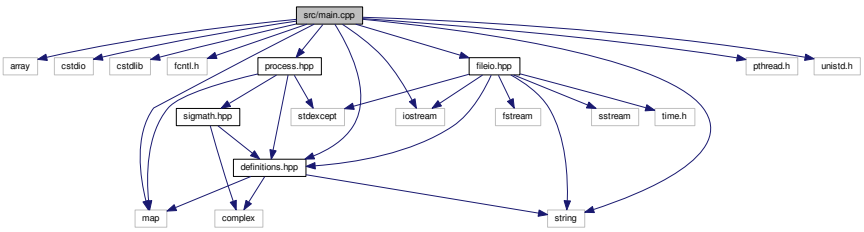
```
00001
00008 #include <fstream>
00009 #include <iostream>
00010 #include <sstream>
00011 #include <string>
00012 #include <time.h>
00013
00014 #include "definitions.hpp"
00015 #include "fileio.hpp"
00016 #include "process.hpp"
00017
00018 using namespace std;
00019 using namespace vaso;
00020
00024 int main() {
00025     string path = PATIENT_PATH + "wizmack, sammy andy.csv";
00026     map<Side, DataParams> laMap = ReadParams(path);
00027     cout <<  laMap[Side::Right].freq << endl;
00028     cout << laMap[Side::Right].noise << endl;
00029
00030     WriteParams(laMap, path);
00031 }
```

## 7.11 src/main.cpp File Reference

contains the main program

```
#include <array>
#include <cstdio>
#include <cstdlib>
#include <fcntl.h>
#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)

### 7.11.1 Detailed Description

contains the main program

#### Author

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

Definition in file main.cpp.

### 7.11.2 Function Documentation

#### 7.11.2.1 int main ( int *argc*, char ** *argv* )

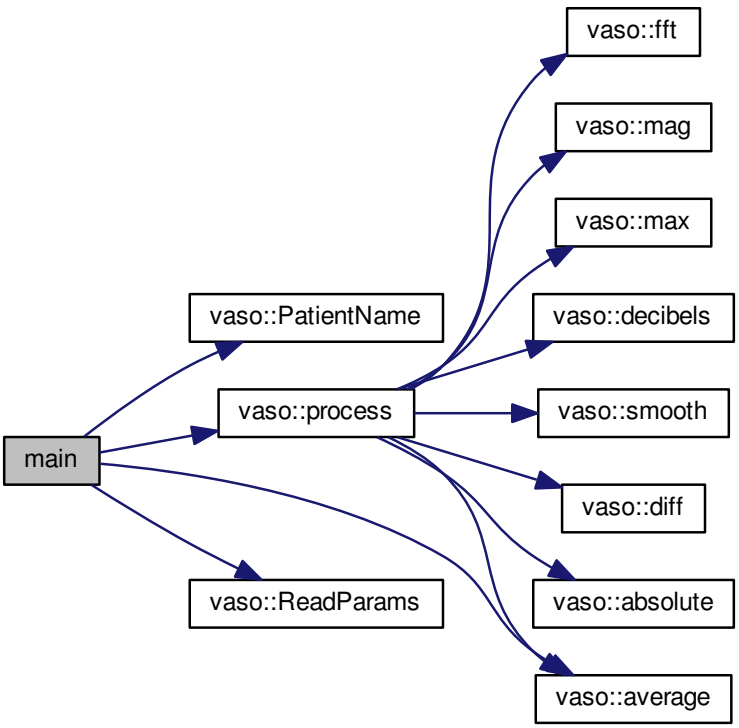
The main program for this project. It will detect vasospasms over a period of days.

Definition at line 29 of file main.cpp.

```
00029                                {
00030     const string recCommand = string("arecord -t raw -d ")
00031         + to_string(DURATION) + string(" -D plughw:1,0 -f FLOAT -q -r ")
00032          + to_string(SAMPLE_FREQ) + string(" ") + TEMP_FILE;
00033     DataParams params[REC_COUNT];
00034
00035     string filename = PatientName();  // generate name for patient's file
00036
00037     // Recorded audio buffer
00038     float32* buffer = (float32*)std::malloc(BUFFER_SIZE);
00039
00040     // Start recording
00041     for(uint8 i = 0; i < REC_COUNT; i++) {
00042         cout << "Press [ ENTER ] to begin analysis for the "
00043             << (i < REC_COUNT / 2 ? "left" : "right") << " side, depth #"
00044             << (((i >= REC_COUNT / 2) ? (i - REC_COUNT / 2) : i) + 1)
00045             << "...";
00046         fflush(stdin);
00047         getchar();  // wait for ENTER to be pressed
00048         cout << "Recording..." << endl;
00049
00050         system(recCommand.c_str());
00051         sleep(DURATION + 1);
00052
00053         int file = open(TEMP_FILE.c_str(), O_RDONLY);
00054         int retRead = read(file, buffer, BUFFER_SIZE);
00055         close(file);
00056         remove(TEMP_FILE.c_str());
00057
00058         if(file < 0 || retRead < BUFFER_SIZE) {
```

```
00059             cerr << "An error occurred reading the doppler audio! "
00060                 "The program will now exit." << endl;
00061             return ERROR;
00062         }
00063
00064         params[i] = process(buffer, SAMPLE_COUNT, SAMPLE_FREQ);
00065         cout << "The analysis is complete." << endl << endl;
00066     }
00067
00068     free(buffer);
00069     map<Side, DataParams> results;
00070     results[Side::Left] = average(params, REC_COUNT / 2);
00071     results[Side::Right] = average(&params[REC_COUNT / 2], REC_COUNT / 2);
00072
00073     cout << "Analysis is complete." << endl << endl;
00074
00075     for(int i = 0; i < 2; i++) {
00076         Side side = (Side)i;
00077         cout << (side == Side::Left ? "[LEFT]" : "[RIGHT]") << endl;
00078         cout << "Drop-off frequency: " << (uint16)(results[side].freq + 0.5)
00079             << " Hz" << endl;
00080         cout << "Average relative noiseband power: "
00081             << (sint16)(results[side].noise - 0.5) << " dB" << endl <<endl;
00082     }
00083
00084     try {
00085         map<Side, DataParams> baseParams = ReadParams(filename);
00086         // TODO: Print results & probable diagnosis
00087
00088     } catch(exception ex) {
00089
00090         // TODO: Write all results to file
00091     }
00092
00093 }
```

Here is the call graph for this function:



## 7.12 main.cpp

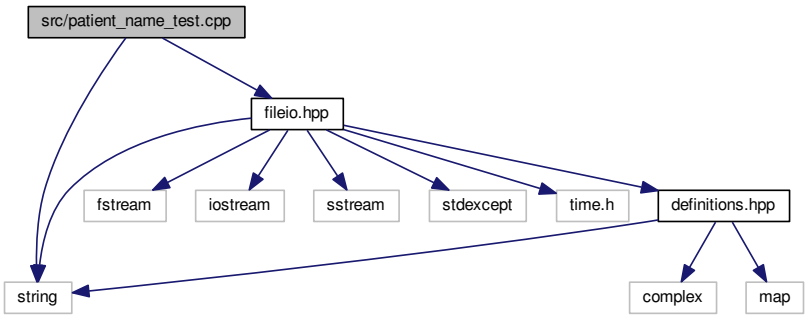
```
00001
00008 #include <array>
00009 #include <cstdio>
00010 #include <cstdlib>
00011 #include <fcntl.h>
00012 #include <iostream>
00013 #include <map>
00014 #include <pthread.h>
00015 #include <string>
00016 #include <unistd.h>
00017
00018 #include "definitions.hpp"
00019 #include "fileio.hpp"
00020 #include "process.hpp"
00021
00022 using namespace std;
00023 using namespace vaso;
00024
00029 int main(int argc, char** argv) {
00030     const string recCommand = string("arecord -t raw -d ")
00031         + to_string(DURATION) + string(" -D plughw:1,0 -f FLOAT -q -r ")
00032          + to_string(SAMPLE_FREQ) + string(" ") + TEMP_FILE;
00033     DataParams params[REC_COUNT];
00034
00035     string filename = PatientName();  // generate name for patient's file
00036
00037     // Recorded audio buffer
00038     float32* buffer = (float32*)std::malloc(BUFFER_SIZE);
00039
00040     // Start recording
00041     for(uint8 i = 0; i < REC_COUNT; i++) {
00042         cout << "Press [ ENTER ] to begin analysis for the "
00043             << (i < REC_COUNT / 2 ? "left" : "right") << " side, depth #"
00044             << (((i >= REC_COUNT / 2) ? (i - REC_COUNT / 2) : i) + 1)
00045             << "...";
00046         fflush(stdin);
00047         getchar();  // wait for ENTER to be pressed
00048         cout << "Recording..." << endl;
00049
00050         system(recCommand.c_str());
00051         sleep(DURATION + 1);
00052
00053         int file = open(TEMP_FILE.c_str(), O_RDONLY);
00054         int retRead = read(file, buffer, BUFFER_SIZE);
00055         close(file);
00056         remove(TEMP_FILE.c_str());
00057
00058         if(file < 0 || retRead < BUFFER_SIZE) {
00059             cerr << "An error occurred reading the doppler audio! "
00060                 "The program will now exit." << endl;
00061             return ERROR;
00062         }
00063
00064         params[i] = process(buffer, SAMPLE_COUNT, SAMPLE_FREQ);
00065         cout << "The analysis is complete." << endl << endl;
00066     }
00067
00068     free(buffer);
00069     map<Side, DataParams> results;
00070     results[Side::Left] = average(params, REC_COUNT / 2);
00071     results[Side::Right] = average(&params[REC_COUNT / 2], REC_COUNT / 2);
00072
00073     cout << "Analysis is complete." << endl << endl;
00074
00075     for(int i = 0; i < 2; i++) {
00076         Side side = (Side)i;
00077         cout << (side == Side::Left ? "[LEFT]" : "[RIGHT]") << endl;
00078         cout << "Drop-off frequency: " << (uint16)(results[side].freq + 0.5)
00079             << " Hz" << endl;
00080         cout << "Average relative noiseband power: "
00081             << (sint16)(results[side].noise - 0.5) << " dB" << endl <<endl;
00082     }
00083
00084     try {
00085         map<Side, DataParams> baseParams = ReadParams(filename);
00086         // TODO: Print results & probable diagnosis
00087
00088     } catch(exception ex) {
00089
00090         // TODO: Write all results to file
00091     }
00092
00093 }
```

## 7.13 src/patient_name_test.cpp File Reference

Contains a program to test the PatientName() function.

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

Include dependency graph for patient_name_test.cpp:




### Functions

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

### 7.13.1 Detailed Description

Contains a program to test the PatientName() function.

#### Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file patient_name_test.cpp.

### 7.13.2 Function Documentation

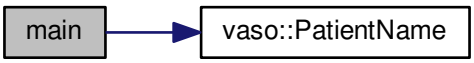
#### 7.13.2.1 int main ( int *argc*, char ** *argv* )

Tests the PatientName() function from fileio.hpp.

Definition at line 17 of file patient_name_test.cpp.

```
00017                                {
00018     string filename = PatientName();
00019     cout << filename;
00020 }
```

Here is the call graph for this function:




## 7.14 patient_name_test.cpp

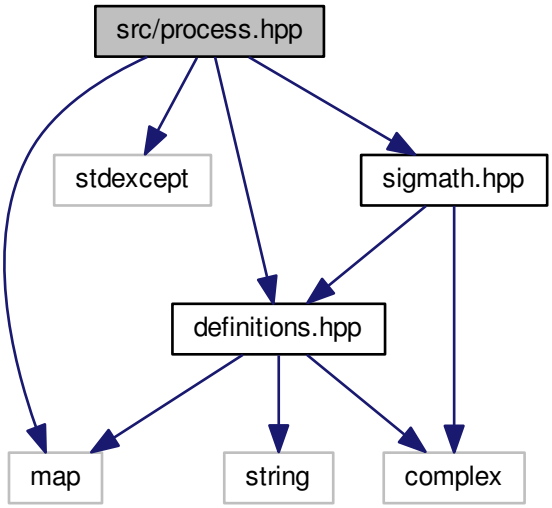
```
00001
00007 #include <string>
00008
00009 #include "fileio.hpp"
00010
00011 using namespace std;
00012 using namespace vaso;
00013
00017 int main(int argc, char** argv) {
00018     string filename = PatientName();
00019     cout << filename;
00020 }
```

## 7.15 src/process.hpp File Reference

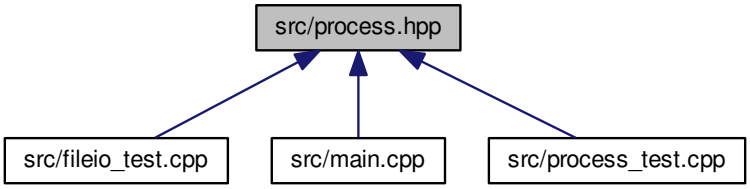
Contains functions related to the program's threaded processing of audio data.

```
#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

## Functions

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

### 7.15.1 Detailed Description

Contains functions related to the program's threaded processing of audio data.

**Author**

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file process.hpp.

## 7.16 process.hpp

```
00001
00008 #ifndef process_H
00009 #define process_H
00010
00011 #include <map>
00012 #include <stdexcept>
00013
00014 #include "definitions.hpp"
00015 #include "sigmath.hpp"
00016
00017 namespace vaso {
00048     DataParams process(float32* data, uint32 size,
      float32 samplingRate) {
00049         if((size & (size - 1) != 0) || size < 2) {
00050             throw std::invalid_argument(
00051                     "The number of samples is not a power of two!");
00052         }
00053
00054         // declare function-scoped variables
00055         uint32 freqSize = size / 2;
00056         cfloat32* cdata = (cfloat32*)std::malloc(size * sizeof(
      cfloat32));
00057         float32* fdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00058         float32* origdata = (float32*)std::malloc(freqSize * sizeof(
      float32));
00059
00060         // convert data to complex numbers for fft()
00061         for(uint32 i = 0; i < size; i++) {
00062             cdata[i] = data[i];
00063         }
00064
00065         // find frequency spectrum in relative decibels
00066         fft(cdata, size);
00067         mag(cdata, fdata, freqSize);
00068         Maximum maximum = max(fdata, freqSize);
00069
00070         for(uint32 i = 0; i < freqSize; i++) {
00071             fdata[i] /= maximum.value;
00072         }
00073
00074         decibels(fdata, freqSize);
00075
00076         for(uint32 i = 0; i < freqSize; i++) {
00077             origdata[i] = fdata[i];
00078         }
00079
00080         /*
00081          * Run spectrum values through moving-average filter to smooth the
00082          * curve and make it easier to determine the derivative.
00083          */
00084         smooth(fdata, freqSize, 20);
00085
00086         /*
00087          * Find the derivative of the smoothed spectrum. Bote that both this
00088          * filter and the previous are necessary to the algorithm.
00089          */
00090         diff(fdata, freqSize);
00091         smooth(fdata, freqSize, 100);
00092         absolute(fdata, freqSize);
00093
00094         // find the parameters of this specific recording
00095         uint16 offset = 1000;
00096         absolute(&fdata[offset], freqSize - offset);
00097         maximum = max(&fdata[offset], freqSize - offset);
00098         uint32 index = maximum.index + offset;
00099
00100         DataParams params;
00101         params.freq = index * (float)SAMPLE_FREQ / freqSize / 2;
00102         params.noise = average(&origdata[index + offset],
00103                 freqSize - offset - index);
00104
00105         free(cdata);
00106         free(fdata);
```

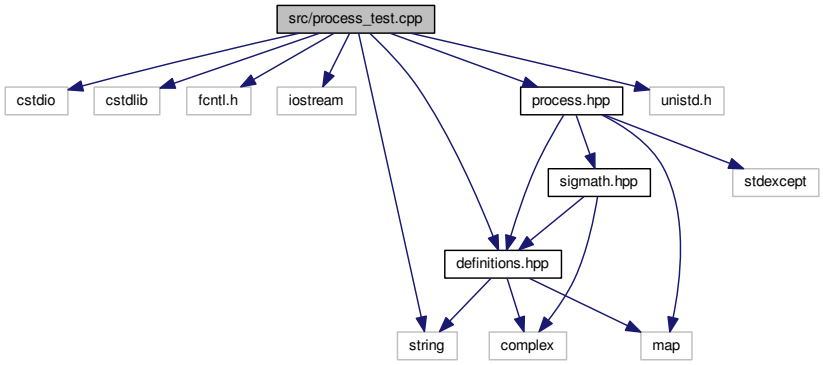
```
00107
00108         return params;
00109
00110     }
00111 }
00112
00113 #endif
```

## 7.17 src/process_test.cpp File Reference

Contains a program to test the process() function.

```
#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)

### 7.17.1 Detailed Description

Contains a program to test the process() function.

#### Author

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

Definition in file process_test.cpp.

### 7.17.2 Macro Definition Documentation

#### 7.17.2.1 #define COUNT 131072

Definition at line 18 of file process_test.cpp.

### 7.17.3 Function Documentation

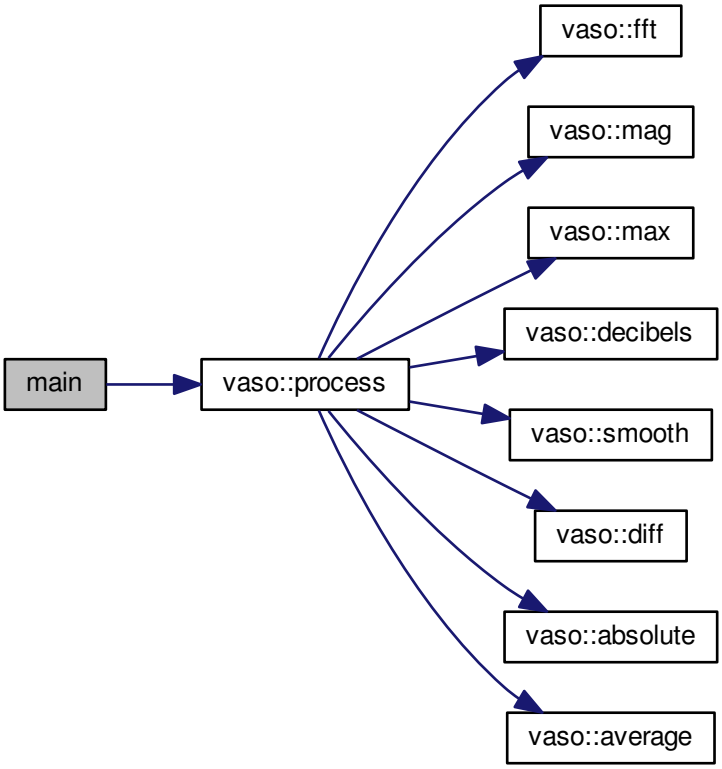
#### 7.17.3.1 int main ( int *argc*, char ** *argv* )

Tests the process() function from process.hpp.

Definition at line 26 of file process_test.cpp.

```
00026                                {
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 }
```

Here is the call graph for this function:




## 7.18 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;
```

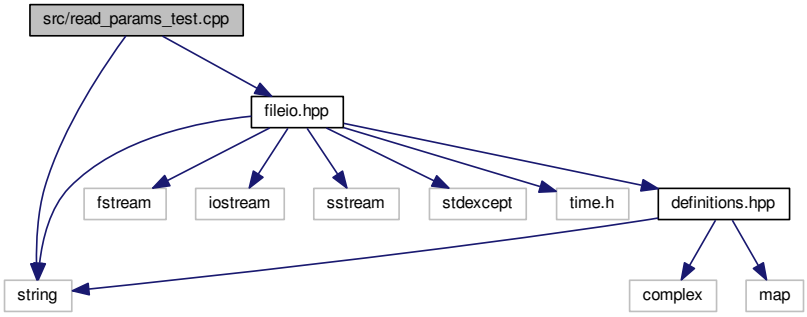
```
00040     }
00041
00042     close(file);
00043
00044     DataParams params = process(buffer, COUNT, SAMPLE_FREQ);
00045     free(buffer);
00046     cout << "Cutoff: " << params.freq << endl;
00047     cout << "Noise: " << params.noise << endl;
00048 }
```

## 7.19 src/read_params_test.cpp File Reference

Contains a program test the PatientName() function.

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

Include dependency graph for read_params_test.cpp:




### Functions

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

### 7.19.1 Detailed Description

Contains a program test the PatientName() function.

#### Author

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file read_params_test.cpp.

### 7.19.2 Function Documentation

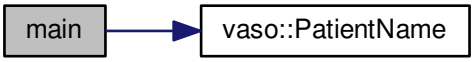
#### 7.19.2.1 int main ( int *argc*, char ** *argv* )

Tests the PatientName() function in fileio.hpp.

Definition at line 17 of file read_params_test.cpp.

```
00017                                {
00018     string filename = PatientName();
00019     cout << filename;
00020 }
```

Here is the call graph for this function:




## 7.20 read_params_test.cpp

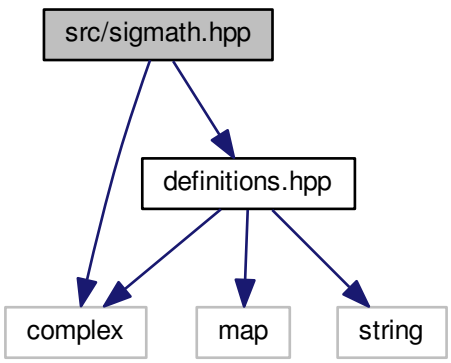
```
00001
00007 #include <string>
00008
00009 #include "fileio.hpp"
00010
00011 using namespace std;
00012 using namespace vaso;
00013
00017 int main(int argc, char** argv) {
00018     string filename = PatientName();
00019     cout << filename;
00020 }
```

## 7.21 src/sigmath.hpp File Reference

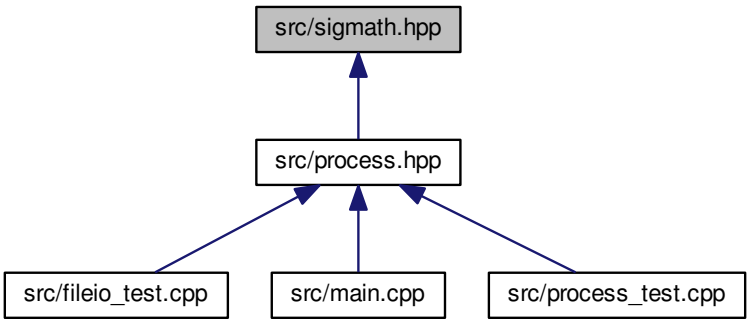
contains the functions necessary to perform the mathematical operations required by this program

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

Include dependency graph for sigmath.hpp:



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




### Namespaces

- vaso

### Functions

- void vaso::absolute (float32 *data, uint32 size)
- float32 vaso::average (float32 *data, uint32 size)
- DataParams vaso::average (DataParams *params, uint8 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)

### 7.21.1 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.

## 7.22 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
00062     void decibels(float32* data, uint32 size);
00063
00072     void diff(float32* data, uint32 size);
00073
00085     void fft(cfloat32* data, uint32 size);
00086
00096     void mag(cfloat32* orig, float32* newmags, uint32 size);
00097
00107     Maximum max(float32* data, uint32 size);
00108
00119     void smooth(float32* data, uint32 size, uint16 order);
00120
00121     // DEFINITIONS
00122
00123     void absolute(float32* data, uint32 size) {
00124         for(uint32 i = 0; i < size; i++) {
00125             data[i] = fabsf(data[i]);
00126         }
00127     }
00128
00129     float32 average(float32* data, uint32 size) {
00130         float32 ave;
00131
00132         for(uint32 i = 0; i < size; i++) {
00133             ave += data[i];
00134         }
00135
00136         ave = ave / size;
00137         return ave;
00138     }
00139
00140     DataParams average(DataParams* params, uint8 size) {
00141         DataParams ave;
00142
00143         for(uint8 i = 0; i < size; i++) {
00144             //freq is an attribute. this is how to add structure attributes
00145             ave.freq += params[i].freq;
00146             ave.noise += params[i].noise;
00147         }
00148
00149         ave.freq /= size;
00150         ave.noise /= size;
00151         return ave;
00152     }
00153
00154     void decibels(float32* data, uint32 size) {
00155         for(uint32 i = 0; i < size; i++) {
00156             data[i] = 20 * log10(data[i]);
00157         }
00158     }
00159
00160     void diff(float32* data, uint32 size) {
00161         float32 temp[size];
00162         temp[0] = 0;
00163
00164         for(uint32 i = 1; i < size; i++) {
00165             temp[i] = data[i] - data[i-1];
00166         }
00167
00168         for(uint32 i = 0; i < size; i++) {
00169             data[i] = temp[i];
00170         }
00171     }
00172
00173     void fft(cfloat32* data, uint32 size) {
00174         // DFT
00175         uint32 k = size;
00176         uint32 n;
00177         float32 thetaT = M_PI / size;
00178         cfloat32 phiT(cos(thetaT), sin(thetaT));
00179         cfloat32 T;
00180
00181         while(k > 1) {
00182             n = k;
00183             k >>= 1;
00184             phiT = phiT * phiT;
00185             T = 1.0L;
00186
```

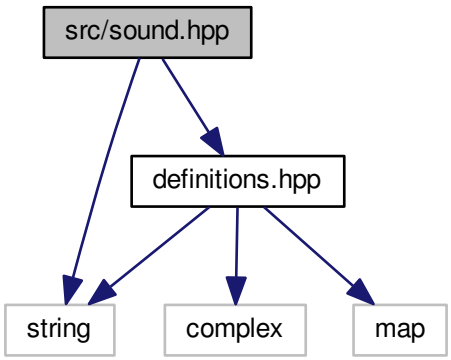
```
00187             for(uint32 l = 0; l < k; l++) {
00188                 for(uint32 a = l; a < size; a += n) {
00189                     uint32 b = a + k;
00190                     cfloat32 t = data[a] - data[b];
00191                     data[a] += data[b];
00192                     data[b] = t * T;
00193                 }
00194 
00195                 T *= phiT;
00196             }
00197         }
00198 
00199         // Decimate
00200         uint32 m = (uint32)log2(size);
00201 
00202         for(uint32 a = 0; a < size; a++) {
00203             uint32 b = a;
00204 
00205             // Reverse bits
00206             b = (((b & 0xaaaaaaaa) >> 1) | ((b & 0x55555555) << 1));
00207             b = (((b & 0xcccccccc) >> 2) | ((b & 0x33333333) << 2));
00208             b = (((b & 0xf0f0f0f0) >> 4) | ((b & 0x0f0f0f0f) << 4));
00209             b = (((b & 0xff00ff00) >> 8) | ((b & 0x00ff00ff) << 8));
00210             b = ((b >> 16) | (b << 16)) >> (32 - m);
00211 
00212             if (b > a)
00213             {
00214                 cfloat32 t = data[a];
00215                 data[a] = data[b];
00216                 data[b] = t;
00217             }
00218         }
00219     }
00220 
00221     void mag(cfloat32* orig, float32* newmags, uint32 size) {
00222         //loop to run throught the length of array orig
00223         for(uint32 n = 0; n < size; n++) {
00224             /*
00225              * abs should calculate the magnitude of complex array elements.
00226              * saves to new array
00227              */
00228             newmags[n] = std::abs(orig[n]);
00229         }
00230     }
00231 
00232     Maximum max(float32* data, uint32 size) {
00233         Maximum m;
00234 
00235         //loop to run through the length of array data
00236         for (uint32 i = 0; i < size; i++) {
00237             /*
00238              * when value at data[i] is above max.value,
00239              * sets max.value equal to data[i] and max.index equal to i
00240              */
00241             if (data[i] > m.value) {
00242                 m.value = data[i];
00243                 m.index = i;
00244             }
00245         }
00246 
00247         return m;
00248     }
00249 
00250     void smooth(float32* data, uint32 size, uint16 order) {
00251         float32 coeff = 1 / (float32)order;
00252         float32 temp[size];
00253 
00254         for(uint32 i = 0; i < size; i++) {
00255             temp[i] = 0;
00256 
00257             for(uint16 j = 0; j < order && j <= i; j++) {
00258                 temp[i] += data[i - j];
00259             }
00260 
00261             temp[i] *= coeff;
00262         }
00263 
00264         for(uint32 i = 0; i < size; i++) {
00265             data[i] = temp[i];
00266         }
00267     }
00268 }
00269 
00270 #endif
```

## 7.23 src/sound.hpp File Reference

contains the function(s) relating to sound

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

Include dependency graph for sound.hpp:




### Namespaces

- vaso

### Functions

- void vaso::play (auto filename)

### 7.23.1 Detailed Description

contains the function(s) relating to sound

**Author**

Samuel Andrew Wisner, awisner94@gmail.com

Definition in file sound.hpp.

## 7.24 sound.hpp

```
00001
00007 #ifndef sound_H
00008 #define sound_H
00009
00010 #include <string>
00011
00012 #include "definitions.hpp"
00013
00014 namespace vaso {
00020     void play(auto filename) {
```

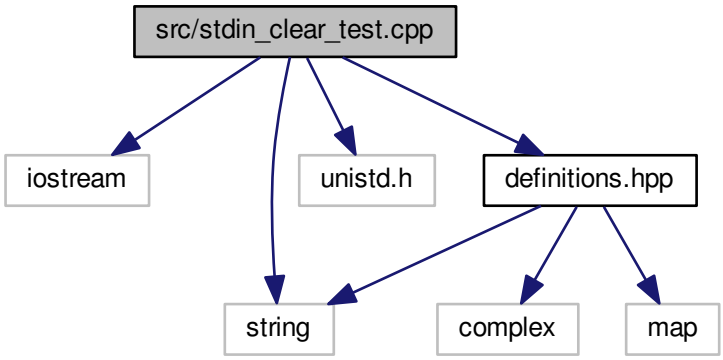
```
00021
00022     }
00023 }
00024
00025 #endif
```

## 7.25 src/stdin_clear_test.cpp File Reference

Contains a program to test clearing the stdin buffer.

```
#include <iostream>
#include <string>
#include <unistd.h>
#include "definitions.hpp"
```

Include dependency graph for stdin_clear_test.cpp:




### Macros

- #define COUNT 80

### Functions

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

### 7.25.1 Detailed Description

Contains a program to test clearing the stdin buffer.

**Author**

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

Definition in file stdin_clear_test.cpp.

### 7.25.2 Macro Definition Documentation

#### 7.25.2.1 #define COUNT 80

Definition at line 14 of file stdin_clear_test.cpp.

### 7.25.3 Function Documentation

#### 7.25.3.1 int main ( int *argc,* char ** *argv* )

Tests the ability to clear the stdin buffer.

Definition at line 22 of file stdin_clear_test.cpp.

```
00022                                {
00023     char text1[COUNT];
00024     char text2[COUNT];
00025
00026     cout << "Enter text to ignore: ";
00027     cout.flush();
00028     read(STDIN_FILENO, &text1, COUNT);
00029     fflush(stdin);
00030     cout << endl << "Enter text to print: ";
00031     cout.flush();
00032     read(STDIN_FILENO, &text2, COUNT);
00033     cout << endl << "In buffer: " << text2 << endl;
00034 }
```

## 7.26 stdin_clear_test.cpp

```
00001
00008 #include <iostream>
00009 #include <string>
00010 #include <unistd.h>
00011
00012 #include "definitions.hpp"
00013
00014 #define COUNT 80
00015
00016 using namespace std;
00017 using namespace vaso;
00018
00022 int main(int argc, char** argv) {
00023     char text1[COUNT];
00024     char text2[COUNT];
00025
00026     cout << "Enter text to ignore: ";
00027     cout.flush();
00028     read(STDIN_FILENO, &text1, COUNT);
00029     fflush(stdin);
00030     cout << endl << "Enter text to print: ";
00031     cout.flush();
00032     read(STDIN_FILENO, &text2, COUNT);
00033     cout << endl << "In buffer: " << text2 << endl;
00034 }
```

# Index