# processEEG

Generated by Doxygen 1.8.10

Mon Jan 2 2017 15:39:08

# Contents

# Chapter 1

# Research

# 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 documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 COLOUR Struct Reference

the RGB representation of every pixel

### Public Attributes

- double **r**
- double **g**
- double **b**

### 4.1.1 Detailed Description

the RGB representation of every pixel

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

- /Users/vinay/Google Drive/Science/Research/Morlets/Plot.cc

## 4.2 edf_annotation_struct Struct Reference

### Public Attributes

- long long **onset**
- char **duration** [16]
- char **annotation** [EDFLIB_MAX_ANNOTATION_LEN+1]

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

- /Users/vinay/Google Drive/Science/Research/include/edflib.h

## 4.3 edf_hdr_struct Struct Reference

### Public Attributes

- int **handle**
- int **filetype**

- int **edfsignals**
- long long **file_duration**
- int **startdate_day**
- int **startdate_month**
- int **startdate_year**
- long long **starttime_subsecond**
- int **starttime_second**
- int **starttime_minute**
- int **starttime_hour**
- char **patient** [81]
- char **recording** [81]
- char **patientcode** [81]
- char **gender** [16]
- char **birthdate** [16]
- char **patient_name** [81]
- char **patient_additional** [81]
- char **admincode** [81]
- char **technician** [81]
- char **equipment** [81]
- char **recording_additional** [81]
- long long **datarecord_duration**
- long long **datarecords_in_file**
- long long **annotations_in_file**
- struct edf_param_struct **signalparam** [EDFLIB_MAXSIGNALS]

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

- /Users/vinay/Google Drive/Science/Research/include/edflib.h

## 4.4 edf_param_struct Struct Reference

### Public Attributes

- char **label** [17]
- long long **smp_in_file**
- double **phys_max**
- double **phys_min**
- int **dig_max**
- int **dig_min**
- int **smp_in_datarecord**
- char **physdimension** [9]
- char **prefilter** [81]
- char **transducer** [81]

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

- /Users/vinay/Google Drive/Science/Research/include/edflib.h

## 4.5 edfhdrblock Struct Reference

### Public Attributes

- FILE ∗ **file_hdl**
- char **path** [1024]
- int **writemode**
- char **version** [32]
- char **patient** [81]
- char **recording** [81]
- char **plus_patientcode** [81]
- char **plus_gender** [16]
- char **plus_birthdate** [16]
- char **plus_patient_name** [81]
- char **plus_patient_additional** [81]
- char **plus_startdate** [16]
- char **plus_admincode** [81]
- char **plus_technician** [81]
- char **plus_equipment** [81]
- char **plus_recording_additional** [81]
- long long **l_starttime**
- int **startdate_day**
- int **startdate_month**
- int **startdate_year**
- int **starttime_second**
- int **starttime_minute**
- int **starttime_hour**
- char **reserved** [45]
- int **hdrsize**
- int **edfsignals**
- long long **datarecords**
- int **recordsize**
- int **annot_ch** [EDFLIB_MAXSIGNALS]
- int **nr_annot_chns**
- int **mapped_signals** [EDFLIB_MAXSIGNALS]
- int **edf**
- int **edfplus**
- int **bdf**
- int **bdfplus**
- int **discontinuous**
- int **signal_write_sequence_pos**
- long long **starttime_offset**
- double **data_record_duration**
- long long **long_data_record_duration**
- int **annots_in_file**
- int **annotlist_sz**
- int **total_annot_bytes**
- int **eq_sf**
- struct edfparamblock ∗ **edfparam**

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

- /Users/vinay/Google Drive/Science/Research/src/edflib.c

## 4.6 edfparamblock Struct Reference

### Public Attributes

- char **label** [17]
- char **transducer** [81]
- char **physdimension** [9]
- double **phys_min**
- double **phys_max**
- int **dig_min**
- int **dig_max**
- char **prefilter** [81]
- int **smp_per_record**
- char **reserved** [33]
- double **offset**
- int **buf_offset**
- double **bitvalue**
- int **annotation**
- long long **sample_pntr**

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

- /Users/vinay/Google Drive/Science/Research/src/edflib.c

## 4.7 pngwriter Class Reference

### Public Member Functions

- **pngwriter** (const pngwriter &rhs)
- **pngwriter** (int width, int height, int backgroundcolour, char ∗filename)
- **pngwriter** (int width, int height, double backgroundcolour, char ∗filename)
- **pngwriter** (int width, int height, int backgroundcolour, const char ∗filename)
- **pngwriter** (int width, int height, double backgroundcolour, const char ∗filename)
- pngwriter & **operator=** (const pngwriter &rhs)
- void **plot** (int x, int y, int red, int green, int blue)
- void **plot** (int x, int y, double red, double green, double blue)
- void **plotHSV** (int x, int y, double hue, double saturation, double value)
- void **plotHSV** (int x, int y, int hue, int saturation, int value)
- int **read** (int x, int y, int colour)
- int **read** (int x, int y)
- double **dread** (int x, int y, int colour)
- double **dread** (int x, int y)
- int **readHSV** (int x, int y, int colour)
- double **dreadHSV** (int x, int y, int colour)
- void **clear** (void)
- void **close** (void)
- void **pngwriter_rename** (char ∗newname)
- void **pngwriter_rename** (const char ∗newname)
- void **pngwriter_rename** (long unsigned int index)
- void **line** (int xfrom, int yfrom, int xto, int yto, int red, int green, int blue)
- void **line** (int xfrom, int yfrom, int xto, int yto, double red, double green, double blue)
- void **triangle** (int x1, int y1, int x2, int y2, int x3, int y3, int red, int green, int blue)
- void **triangle** (int x1, int y1, int x2, int y2, int x3, int y3, double red, double green, double blue)

- void **square** (int xfrom, int yfrom, int xto, int yto, int red, int green, int blue)
- void **square** (int xfrom, int yfrom, int xto, int yto, double red, double green, double blue)
- void **filledsquare** (int xfrom, int yfrom, int xto, int yto, int red, int green, int blue)
- void **filledsquare** (int xfrom, int yfrom, int xto, int yto, double red, double green, double blue)
- void **circle** (int xcentre, int ycentre, int radius, int red, int green, int blue)
- void **circle** (int xcentre, int ycentre, int radius, double red, double green, double blue)
- void **filledcircle** (int xcentre, int ycentre, int radius, int red, int green, int blue)
- void **filledcircle** (int xcentre, int ycentre, int radius, double red, double green, double blue)
- void **readfromfile** (char ∗name)
- void **readfromfile** (const char ∗name)
- int **getheight** (void)
- int **getwidth** (void)
- void **setcompressionlevel** (int level)
- int **getbitdepth** (void)
- int **getcolortype** (void)
- void **setgamma** (double gamma)
- double **getgamma** (void)
- void **bezier** (int startPtX, int startPtY, int startControlX, int startControlY, int endPtX, int endPtY, int end←ControlX, int endControlY, double red, double green, double blue)
- void **bezier** (int startPtX, int startPtY, int startControlX, int startControlY, int endPtX, int endPtY, int end←ControlX, int endControlY, int red, int green, int blue)
- void **settext** (char ∗title, char ∗author, char ∗description, char ∗software)
- void **settext** (const char ∗title, const char ∗author, const char ∗description, const char ∗software)
- void **write_png** (void)
- void **plot_text** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, double red, double green, double blue)
- void **plot_text** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, int red, int green, int blue)
- void **plot_text_utf8** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, double red, double green, double blue)
- void **plot_text_utf8** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, int red, int green, int blue)
- int **bilinear_interpolation_read** (double x, double y, int colour)
- double **bilinear_interpolation_dread** (double x, double y, int colour)
- void **plot_blend** (int x, int y, double opacity, int red, int green, int blue)
- void **plot_blend** (int x, int y, double opacity, double red, double green, double blue)
- void **invert** (void)
- void **resize** (int width, int height)
- void **boundary_fill** (int xstart, int ystart, double boundary_red, double boundary_green, double boundary_←blue, double fill_red, double fill_green, double fill_blue)
- void **boundary_fill** (int xstart, int ystart, int boundary_red, int boundary_green, int boundary_blue, int fill_red, int fill_green, int fill_blue)
- void **flood_fill** (int xstart, int ystart, double fill_red, double fill_green, double fill_blue)
- void **flood_fill** (int xstart, int ystart, int fill_red, int fill_green, int fill_blue)
- void **polygon** (int ∗points, int number_of_points, double red, double green, double blue)
- void **polygon** (int ∗points, int number_of_points, int red, int green, int blue)
- void **plotCMYK** (int x, int y, double cyan, double magenta, double yellow, double black)
- void **plotCMYK** (int x, int y, int cyan, int magenta, int yellow, int black)
- double **dreadCMYK** (int x, int y, int colour)
- int **readCMYK** (int x, int y, int colour)
- void **scale_k** (double k)
- void **scale_kxky** (double kx, double ky)
- void **scale_wh** (int finalwidth, int finalheight)
- void **plotHSV_blend** (int x, int y, double opacity, double hue, double saturation, double value)
- void **plotHSV_blend** (int x, int y, double opacity, int hue, int saturation, int value)

- void **line_blend** (int xfrom, int yfrom, int xto, int yto, double opacity, int red, int green, int blue)
- void **line_blend** (int xfrom, int yfrom, int xto, int yto, double opacity, double red, double green, double blue)
- void **square_blend** (int xfrom, int yfrom, int xto, int yto, double opacity, int red, int green, int blue)
- void **square_blend** (int xfrom, int yfrom, int xto, int yto, double opacity, double red, double green, double blue)
- void **filledsquare_blend** (int xfrom, int yfrom, int xto, int yto, double opacity, int red, int green, int blue)
- void **filledsquare_blend** (int xfrom, int yfrom, int xto, int yto, double opacity, double red, double green, double blue)
- void **circle_blend** (int xcentre, int ycentre, int radius, double opacity, int red, int green, int blue)
- void **circle_blend** (int xcentre, int ycentre, int radius, double opacity, double red, double green, double blue)
- void **filledcircle_blend** (int xcentre, int ycentre, int radius, double opacity, int red, int green, int blue)
- void **filledcircle_blend** (int xcentre, int ycentre, int radius, double opacity, double red, double green, double blue)
- void **bezier_blend** (int startPtX, int startPtY, int startControlX, int startControlY, int endPtX, int endPtY, int endControlX, int endControlY, double opacity, double red, double green, double blue)
- void **bezier_blend** (int startPtX, int startPtY, int startControlX, int startControlY, int endPtX, int endPtY, int endControlX, int endControlY, double opacity, int red, int green, int blue)
- void **plot_text_blend** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, double opacity, double red, double green, double blue)
- void **plot_text_blend** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, double opacity, int red, int green, int blue)
- void **plot_text_utf8_blend** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, double opacity, double red, double green, double blue)
- void **plot_text_utf8_blend** (char ∗face_path, int fontsize, int x_start, int y_start, double angle, char ∗text, double opacity, int red, int green, int blue)
- void **boundary_fill_blend** (int xstart, int ystart, double opacity, double boundary_red, double boundary_↩ green, double boundary_blue, double fill_red, double fill_green, double fill_blue)
- void **boundary_fill_blend** (int xstart, int ystart, double opacity, int boundary_red, int boundary_green, int boundary_blue, int fill_red, int fill_green, int fill_blue)
- void **flood_fill_blend** (int xstart, int ystart, double opacity, double fill_red, double fill_green, double fill_blue)
- void **flood_fill_blend** (int xstart, int ystart, double opacity, int fill_red, int fill_green, int fill_blue)
- void **polygon_blend** (int ∗points, int number_of_points, double opacity, double red, double green, double blue)
- void **polygon_blend** (int ∗points, int number_of_points, double opacity, int red, int green, int blue)
- void **plotCMYK_blend** (int x, int y, double opacity, double cyan, double magenta, double yellow, double black)
- void **plotCMYK_blend** (int x, int y, double opacity, int cyan, int magenta, int yellow, int black)
- void **laplacian** (double k, double offset)
- void **filledtriangle** (int x1, int y1, int x2, int y2, int x3, int y3, int red, int green, int blue)
- void **filledtriangle** (int x1, int y1, int x2, int y2, int x3, int y3, double red, double green, double blue)
- void **filledtriangle_blend** (int x1, int y1, int x2, int y2, int x3, int y3, double opacity, int red, int green, int blue)
- void **filledtriangle_blend** (int x1, int y1, int x2, int y2, int x3, int y3, double opacity, double red, double green, double blue)
- void **arrow** (int x1, int y1, int x2, int y2, int size, double head_angle, double red, double green, double blue)
- void **arrow** (int x1, int y1, int x2, int y2, int size, double head_angle, int red, int green, int blue)
- void **filledarrow** (int x1, int y1, int x2, int y2, int size, double head_angle, double red, double green, double blue)
- void **filledarrow** (int x1, int y1, int x2, int y2, int size, double head_angle, int red, int green, int blue)
- void **cross** (int x, int y, int xwidth, int yheight, double red, double green, double blue)
- void **cross** (int x, int y, int xwidth, int yheight, int red, int green, int blue)
- void **maltesecross** (int x, int y, int xwidth, int yheight, int x_bar_height, int y_bar_width, double red, double green, double blue)
- void **maltesecross** (int x, int y, int xwidth, int yheight, int x_bar_height, int y_bar_width, int red, int green, int blue)
- void **filleddiamond** (int x, int y, int width, int height, int red, int green, int blue)
- void **diamond** (int x, int y, int width, int height, int red, int green, int blue)
- void **filleddiamond** (int x, int y, int width, int height, double red, double green, double blue)
- void **diamond** (int x, int y, int width, int height, double red, double green, double blue)
- int **get_text_width** (char ∗face_path, int fontsize, char ∗text)
- int **get_text_width_utf8** (char ∗face_path, int fontsize, char ∗text)

### Static Public Member Functions

- static double **version** (void)

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

- /Users/vinay/Google Drive/Science/Research/include/pngwriter.h

## 4.8 RANGE Struct Reference

The lower and upper bound of the data matrix.

### Public Attributes

- double **minimum**
- double **maximum**

### 4.8.1 Detailed Description

The lower and upper bound of the data matrix.

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

- /Users/vinay/Google Drive/Science/Research/Morlets/Plot.cc

# Chapter 5

# File Documentation

## 5.1 /Users/vinay/Google Drive/Science/Research/include/processEEG.h File Reference

The files needed to open and process BDF Files.

```
#include "edflib.h"
#include "wavelet.h"
```

### Macros

- #define PRE_EVENT_TIME 1.0

  *The amount of seconds before the stimulus.*
- #define POST_EVENT_TIME 2.0

  *The amount of seconds after the stimulus.*
- #define **MAXIMUM_TRIGGERS** 1000000

### Functions

- int OpenFile (const char *fileName, struct edf_hdr_struct *header)

  *Openes a .BDF file and allocates it to an edf_hdr_struct.*
- long long FindTriggers (const int *statusInput, const long long numberOfElements, long long *outputBuffer)

  *This function should take an array input and return the rising and falling edges of the triggers.*
- int FilterTriggers (const int code, const int button, const int numberOfRecords, const long long *triggerList, const int *readBuffer, int *outputBuffer)

  *Filteres the triggers coming in, and finds the specified events.*

### 5.1.1 Detailed Description

The files needed to open and process BDF Files.

### 5.1.2 Macro Definition Documentation

#### 5.1.2.1 #define POST_EVENT_TIME 2.0

The amount of seconds after the stimulus.

POST_EVENT_TIME

#### 5.1.2.2 #define PRE_EVENT_TIME 1.0

The amount of seconds before the stimulus.

PRE_EVENT_TIME

### 5.1.3 Function Documentation

#### 5.1.3.1 int FilterTriggers ( const int *code,* const int *button,* const int *numberOfRecords,* const long long ∗ *triggerList,* const int ∗ *readBuffer,* int ∗ *outputBuffer* )

Filteres the triggers coming in, and finds the specified events.

**Parameters**

| | |
|---|---|
| *code* | The code of the trigger list that is needed Possible Inputs: 1, or 2 |
| *button* | The code of the button that is needed Possible Inputs 1, or 2 |
| *triggerList* | The list of all of the possible triggers |
| *readBuffer* | The Status Channel input from the file |
| *outputBuffer* | The buffer that FilterTriggers will populate with the location of the location of the triggers that we're looking for |

**Returns**

counterVariable The number of triggers found.

#### 5.1.3.2 long long FindTriggers ( const int ∗ *statusInput,* const long long *numberOfRecords,* long long ∗ *outputBuffer* )

This function should take an array input and return the rising and falling edges of the triggers.

**Parameters**

| | |
|---|---|
| *statusInput* | The Status Channel Input from the BDF or EDF flie. use the edfread_digital_samples |
| *numberOf←Records* | The size of statusInput |
| *outputBuffer* | a 1 x 2 ∗ MAXIMUM_TRIGGERS long long array with the odd entries being the rising edge and the even entries being the falling edges. |

**Returns**

counterVariable The number of triggers that were found.

#### 5.1.3.3 int OpenFile ( const char ∗ *fileName,* struct **edf_hdr_struct** ∗ *header* )

Openes a .BDF file and allocates it to an edf_hdr_struct.

**Parameters**

| | |
|---|---|
| *fileName* | The name and location of the file to be opened |
| *header* | The pointer to the edf header structure |

**Returns**

0 if file is opened successfully
-1 if there is an error

## 5.2 /Users/vinay/Google Drive/Science/Research/include/wavelet.h File Reference

The supporting header file for generating the Continuous Wavelet Transform.

```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>
#include <math.h>
#include <fftw3.h>
#include <omp.h>
```

### Macros

- #define QUAD_ROOT_PI 0.7511255444649425

  *the quad root of pi ( $\pi^{-0.25}$ ) computed to machine precision*
- #define C_SIGMA 1.0000000000018794

  *A constant needed to compute the Morlet Wavelet precalculated to machine precision.*
- #define K_SIGMA 1.5229979744712628e-08

  *A constant needed to compute the Morlet Wavelet precalculated to machine eps.*
- #define W_0 6.0

  *The fundamental frequency of the Morlet Wavelet.*
- #define W_0_2 36.0

  *The fundamental frequency of the Morlet Wavelet squared.*
- #define D_J 0.125
- #define PAD_FLAG 1

  *The type of padding specified for the Continuous Wavelet Transform.*
- #define **FS** 2048
- #define DT 1.0/FS

  $\delta t = \frac{1}{f_s}$
- #define S0 2.0 $*$ DT

  *the lowest scale that can be used to compute the CWT* $s_0 = 2\delta t$
- #define **FREQ** 16.0
- #define **DATA_SIZE** 6144
- #define MAX_FREQUENCY 512.0

  *The maximum frequency that will be analyzed.*
- #define MIN_FREQUENCY 0.5

  *The minimum frequency that will be analyzed.*
- #define **MAX_DATA_SIZE** 10000000
- #define **MIN_I** FREQ_TO_SCALE(MAX_FREQUENCY)
- #define **MAX_I** FREQ_TO_SCALE(MIN_FREQUENCY)
- #define FREQ_TO_SCALE(x) floor( ( log2( (W_0) / (S0 $*$ 2 $*$ M_PI $*$ x) ) )/D_J)

  *Converts a given frequency x to a scale, handy for debugging. Note the scale is divided into sub octaves.*
- #define SCALE_TO_FREQ(x) (W_0)/(x $*$ 2 $*$ M_PI)

  *Converts a given scale x to its corrosponding frequency.*
- #define MAGNITUDE(x, y) (x $*$ x) + (y $*$ y)

  *Computes the 2- norm or the x ^ 2 + y ^ 2, of x and y.*

## Functions

- void FillData (double *data)

  *Populates the input data array with a 3 sparse sine waves.*
- void TestCases (double *data, const int flag)

  *Generates a suite of test case data for wavelet analysis.*
- int ReadFile (double data[ ], char filename[ ])
- int WriteFile (const double *data, const double *frequency, const int x, const int y, const char *filename)

  *A function that writes the Wavelet Results to the disk.*
- int WriteDebug (const double *data, const int length, const char *filename)

  *A function that writes a 1 - d matrix into a log file.*
- int ERSP (double *raw_data, double *scales, const int sampling_frequency, const int n, const int J, int const trials, const int padding_type, double *output)
- void **Plot** (double *data, double *periods, int num_x, int num_y)
- double CompleteFourierMorlet (const double w, const double scale)

  *Computes the Morlet Wavelet in the frequency domain.*
- int Wavelet (double *raw_data, double *scales, double sampling_frequency, int n, int J, double *result)

  *A function that computes the Continuous Wavelet Transform for the data given in raw_data.*
- void CleanData (double *data, double n)
- double * GenerateScales (const double minimum_frequency, const double maximum_frequency, const double s_0)

  *This function generates the scales that will be used in the Continuous Wavelet Transform.*
- double * IdentifyFrequencies (double *scales, int count)

  *Compute the corrosponding frequency to scales used.*
- void **Convolute** (double *data, double *conWindow, double *complexWindow, double conSize, double *result, double *complexResult)
- int CalculatePaddingSize (const int array_size, const int pad_flag)

  *Calculates the size that the padded array should be.*
- int Generate_FFTW_Wisdom (int padded_size)

  *Analyzes the size of the FFTW arrays and generates the optimal plan.*

### 5.2.1 Detailed Description

The supporting header file for generating the Continuous Wavelet Transform.

### 5.2.2 Macro Definition Documentation

#### 5.2.2.1 C_SIGMA 1.0000000000018794

A constant needed to compute the Morlet Wavelet precalculated to machine precision.

$$C_\sigma = (1 + e^{-\sigma^2} - 2e^{-\frac{3\sigma^2}{4}})^{-\frac{1}{2}}$$

#### 5.2.2.2 D_J 0.125

The amount of "sub octaves" or sub scales inbetween the major scales that will be used. The lower the number, the higher the resolution of the result.

#### 5.2.2.3 K_SIGMA 1.5229979744712628e-08

A constant needed to compute the Morlet Wavelet precalculated to machine eps.

$$\kappa_\sigma = e^{-\frac{\sigma^2}{2}}$$

#### 5.2.2.4 PAD_FLAG 1

The type of padding specified for the Continuous Wavelet Transform.

The Continuous Wavelet Transform uses the Fast Fourier Transform. It is sometimes efficient to add additional values to the edge of the data array to improve the speed of the FFT. This method is commonly called padding the data array. This variable determins the type of padding that will be used to assist the Fast Fourier Transform. THe padding options are:

0 - No Padding

- The array will be analyzed with no padding.

1 - Zero Padding

- The size of the array will be enlarged to the closest power of two and zeros will be added to the end.

2 - Ramp Padding

- The array will be doubled in size, and the signal will be ramped up and ramped down to gradually.

If none of these are specified, the array is not padded by default.

### 5.2.3 Function Documentation

#### 5.2.3.1 int CalculatePaddingSize ( const int *array_size,* const int *pad_flag* )

Calculates the size that the padded array should be.

**Parameters**

| | |
|---|---|
| *array_size* | The cardinal or size of the signal sample |
| *pad_flag* | The type of padding required: see PAD_FLAG |

**Returns**

paadded_size The cardinal or size that the padded array should be

This function computes the size of the padded array depending on the type of padding specified. It takes the size of the data array, and type of pad, and returns how large the padded array should be.

#### 5.2.3.2 void CleanData ( double ∗ *data,* double *n* )

**Parameters**

| | |
|---|---|
| *data* | An 1 x n array with the data to be cleaned |
| *n* | The size of the data array. |

Takes a 1 x n array and preforms the Z-Score Calculation The array data will be rewritten

#### 5.2.3.3 double CompleteFourierMorlet ( double *w*, double *scale* )

Computes the Morlet Wavelet in the frequency domain.

**Parameters**

| | |
|---|---|
| *w* | |
| *scale* | |

**Returns**

morlet

This function generates the Morlet Wavelet in the frequency domain normalized by the scale.

The formula computed is

$$\hat{\Psi}_\sigma(\omega) = c_\sigma \pi^{-\frac{1}{4}}(e^{-\frac{1}{2}(\sigma-\omega)^2} - \kappa_\sigma e^{-\frac{1}{2}\omega^2})$$

#### 5.2.3.4 int ERSP ( double ∗ *raw_data*, double ∗ *scales*, const int *sampling_frequency*, const int *n*, const int *J*, int const *trials*, const int *padding_type*, double ∗ *output* )

**Parameters**

| | |
|---|---|
| *raw_data* | A trials ∗ n array containing the data to be analyzed |
| *scales* | A 1 x J array of the scales that the wavelets will be analyzed in |
| *sampling_frequency* | The frequency that the data was sampled in |
| *n* | The numer of samples in each data set |
| *J* | The number of scales to be analyzed |
| *trials* | The number of trials conducted for the ERSP |
| *output* | A n x J array with the resultant ERSP from all of the trials. |

**Returns**

0

This function conducts the Event Related Spectral Pertubation of the given data set *raw_data*. It follows the method outlined by the paper: "Single-trial normalization for event-related spectral decomposition reduces sensitivity to noisy trials".

This function uses the Continuous Wavelet Transform to generate the multi-resolution analysis of the given data.

This function is multi-threaded.

This function deals a lot with Fast Fourier Transforms, and can be optimized by using the Generate_FFTW_Wisedom() function. If no wisdom is provided, an approximate FFT algorithm will be used.

The variable raw_data must contain all of the data for each trial.

raw_data, scales, and output must be pre-allocated.

#### 5.2.3.5 void FillData ( double ∗ *data* )

Populates the input data array with a 3 sparse sine waves.

**Parameters**

| | |
|---|---|
| *data* | A 1 - dimentional block of memory that will be overwritten. |

Similar to TestCases() Sine Wave Sample

Sine Wave Sample

Sine Wave Sample

#### 5.2.3.6 int Generate_FFTW_Wisdom ( int *padded_size* )

Analyzes the size of the FFTW arrays and generates the optimal plan.

**Parameters**

| | |
|---|---|
| *padded_size* | The size of the FFT arrays. |

**Returns**

0 If successful
1 if unsuccessful

This function can be used to optimize FFTW. This function will try to find the fastest FFT method based on the size of the array, and will store this information as "FFTW_plan.wise".

This function does not need to be used, but it can significantly improve performance if it is.

#### 5.2.3.7 double $*$ GenerateScales ( const double *minimum_frequency,* const double *maximum_frequency,* const double *s_0* )

This function generates the scales that will be used in the Continuous Wavelet Transform.

**Parameters**

| | |
|---|---|
| *minimum_frequency* | The lowest frequency that must be observed |
| *maximum_frequency* | The higest frequency that must be observed |
| *s_0* | The smallest scale usually it is $2 * \delta t$ |

**Returns**

scales An 1 x n array with the dyadic scales.

This function computes the dyadic scales to be generated to accurately compute the multi resolution analysis of a signal. Given the minimum frequency and the maximum frequency, the function will generate a 1 x n array with the scales necessary scale factors for the Continuous Wavelet Transform

The Scales array will be allocated in this function, so it is wise to deallocate this array after it is used.

#### 5.2.3.8 double $*$ IdentifyFrequencies ( double $*$ *scales,* int *count* )

Compute the corrosponding frequency to scales used.

**Parameters**

| | |
|---|---|
| *scales* | A 1 x count array of the scales used |

| | |
|---|---|
| *count* | The cardinal of the scales array |

**Returns**

frequency The corrosponding frequency for each scale in the scale array

This function computes the corrosponding frequency for each scale provided in the scales array.

It allocates memory and returns the allocated array

It is wise to dealloate this array after use with the free() function.

#### 5.2.3.9 int ReadFile ( double *data[ ]*, char *filename[ ]* )

**Parameters**

| | |
|---|---|
| *A* | pre allocated 1 dimentional array |
| *filename* | The name and location of the file to be opened |

**Returns**

array_size The number of elements that was read

This function opens a file, and reads the input assuming that the file is stored with one value at every line.

#### 5.2.3.10 void TestCases ( double ∗ *data*, int *flag* )

Generates a suite of test case data for wavelet analysis.

**Parameters**

| | |
|---|---|
| *data* | The 1 x n data array to be populated |
| *flag* | The type of test data to be generated |

This function populates the data array with 3 seconds of sample data. The *flag* parameter specifies the type of test data that will be generated

| Flag Type | Output |
|---|---|
| 1 | Impulse at T = 2 seconds |
| 2 | 2 Sine waves at t = 1.5 seconds at FREQ and 2 ∗ FREQ |
| 3 | 2 sine waves at FREQ and 2 ∗ FREQ from t = 0 to 3 s |
| 4 | Single sine wave at t = 1.0 s |
| 5 | sin(x) from t = 0.0 to 3.0 and 2∗sin(x) from t = 1.5 - 2.0 s |
| 6 | sin(x) from t = 0.0 - 3.0 and sin(x - w0) where w0 = 0.005 from t = 1.0 s - 1.5 s |
| 7 | Frequency Sweet from MIN_FREQUENCY to MAX_FREQUENCY |

Table 5.1: TestCases Flags

#### 5.2.3.11 int Wavelet ( double ∗ *raw_data*, double ∗ *scales*, double *sampling_frequency*, int *n*, int *J*, double ∗ *result* )

A function that computes the Continuous Wavelet Transform for the data given in *raw_data*.

**Parameters**

| | |
|---|---|
| *raw_data* | A 1 x n array with the data required |
| *scales* | A 1 x J array with all of the scales for generating the wavelets |
| *sampling_↩ frequency* | The sampling frequency of the given data |
| *n* | The size of the input data |
| *J* | The number of scales that is provided |
| *result* | An n x J array of contiguous memory that stores the result |

This function preforms the Continuous Wavelet Transform using Morlet Wavelets on the data given in raw_data. It stores the result in the result array.

This function only modifies the result array. The arrays must be pre allocated for this function to work.

You can provide the function with scales of your choosing, or one can generate dyadic scales with the Generate↩ Scales() function.

This function is optimized using openmp to allow for multi threading.

#### 5.2.3.12 int WriteDebug ( const double ∗ *data*, const int *length*, const char ∗ *filename* )

A function that writes a 1 - d matrix into a log file.

**Parameters**

| | |
|---|---|
| *data* | A 1 - dimentional data array containing the data to be written |
| *length* | The size of the data array |
| *filename* | THe name of the file to be written |

**Returns**

0 if successful
-1 if unsuccessful

This function writes a 1 - dimentional array to the disk, it's useful when trying to quickly get the results from an array.

#### 5.2.3.13 int WriteFile ( const double ∗ *data*, const double ∗ *period*, const int *x*, const int *y*, const char ∗ *filename* )

A function that writes the Wavelet Results to the disk.

**Parameters**

| | |
|---|---|
| *data* | A x x y array with the data that is going to be written |
| *period* | A 1 x y array with the frequencies that were analyzed |
| *x* | The number of samples in the signal |
| *y* | The number of frequencies analyzed |
| *filename* | The name of the file that will be written |

**Returns**

0 if successful
-1 if unsuccessful

This function will write the resultant data computed by Wavelet() and ERSP() into the disk so that it can be graphed by Gnuplot. One can plot the output of this function using the matrix.gplot file.

## 5.3 /Users/vinay/Google Drive/Science/Research/Morlets/CleanData.cc File Reference

This file contains the code to remove any noise in the input data.

```
#include "processEEG.h"
#include <gsl/gsl_statistics.h>
```

### Functions

- void CleanData (double ∗data, double n)

### 5.3.1 Detailed Description

This file contains the code to remove any noise in the input data.

### 5.3.2 Function Documentation

#### 5.3.2.1 void CleanData ( double ∗ *data,* double *n* )

**Parameters**

| | |
|---|---|
| *data* | An 1 x n array with the data to be cleaned |
| *n* | The size of the data array. |

Takes a 1 x n array and preforms the Z-Score Calculation The array data will be rewritten

## 5.4 /Users/vinay/Google Drive/Science/Research/Morlets/ERSP.cc File Reference

This file contains all of the function required to generate the Event Related Spectral Pertubation of EEG signals.

```
#include "processEEG.h"
#include <gsl/gsl_statistics.h>
```

### Functions

- int RemoveBaseline (double ∗pre_stimulus, double ∗pre_baseline_array, const int n, const int J, const int sampling_frequency, double ∗output)

    *A function that removes the pre stimulus noise found in EEG signals.*
- int FrequencyMultiply (const fftw_complex ∗fft_data, const int data_size, const double scale, const double dw, fftw_complex ∗filter_convolution)

    *Multiples the signal with the wavelet at a specific scale in the frequency domain.*
- int **PopulateDataArray** (double ∗input_data, const int data_size, const int trial_number, const int padded_↩ size, const int padding_type, fftw_complex ∗output_data)
- int ERSP (double ∗raw_data, double ∗scales, const int sampling_frequency, const int n, const int J, int const trials, const int padding_type, double ∗output)
- int Generate_FFTW_Wisdom (int padded_size)

    *Analyzes the size of the FFTW arrays and generates the optimal plan.*

### 5.4.1 Detailed Description

This file contains all of the function required to generate the Event Related Spectral Pertubation of EEG signals.

### 5.4.2 Function Documentation

#### 5.4.2.1 int ERSP ( double * *raw_data*, double * *scales*, const int *sampling_frequency*, const int *n*, const int *J*, int const *trials*, const int *padding_type*, double * *output* )

**Parameters**

| | |
|---|---|
| *raw_data* | A trials * n array containing the data to be analyzed |
| *scales* | A 1 x J array of the scales that the wavelets will be analyzed in |
| *sampling_↩ frequency* | The frequency that the data was sampled in |
| *n* | The numer of samples in each data set |
| *J* | The number of scales to be analyzed |
| *trials* | The number of trials conducted for the ERSP |
| *output* | A n x J array with the resultant ERSP from all of the trials. |

**Returns**

0

This function conducts the Event Related Spectral Pertubation of the given data set *raw_data*. It follows the method outlined by the paper: "Single-trial normalization for event-related spectral decomposition reduces sensitivity to noisy trials".

This function uses the Continuous Wavelet Transform to generate the multi-resolution analysis of the given data.

This function is multi-threaded.

This function deals a lot with Fast Fourier Transforms, and can be optimized by using the Generate_FFTW_↩ Wisedom() function. If no wisdom is provided, an approximate FFT algorithm will be used.

The variable raw_data must contain all of the data for each trial.

raw_data, scales, and output must be pre-allocated.

#### 5.4.2.2 int FrequencyMultiply ( const fftw_complex * *fft_data*, const int *data_size*, const double *scale*, const double *dw*, fftw_complex * *filter_convolution* )

Multiples the signal with the wavelet at a specific scale in the frequency domain.

**Parameters**

| | |
|---|---|
| *fft_data* | A fftw_complex * data_size array with the signal data in the frequency domain. |
| *data_size* | The size of the data array |
| *scale* | THe scale of the wavelet that will be multiplied with the signal array |
| *dw* | THe discrete increment in the frequency domain for the wavelet |
| *filter_convolution* | A fftw_complex * data_size array with the resulted multiplication |

**Returns**

0

This function mutliples the contents of fft_data with with the wavelet specified by the variable *scale*. It stores the result in filter_convolution.

All arrays must be pre allocated.

#### 5.4.2.3 int Generate_FFTW_Wisdom ( int *padded_size* )

Analyzes the size of the FFTW arrays and generates the optimal plan.

**Parameters**

| | |
|---|---|
| *padded_size* | The size of the FFT arrays. |

**Returns**

0 If successful
1 if unsuccessful

This function can be used to optimize FFTW. This function will try to find the fastest FFT method based on the size of the array, and will store this information as "FFTW_plan.wise".

This function does not need to be used, but it can significantly improve performance if it is.

#### 5.4.2.4 int RemoveBaseline ( double ∗ *pre_stimulus*, double ∗ *pre_baseline_array*, const int *n*, const int *J*, const int *m*, double ∗ *output* )

A function that removes the pre stimulus noise found in EEG signals.

**Parameters**

| | |
|---|---|
| *pre_stimulus* | A 1 x m array to store the pre stimulus data |
| *pre_baseline_array* | An n x J array of the data that must be modified |
| *n* | The number of samples in the entire data array |
| *J* | The number of scales that were used |
| *m* | The size of the array before the stimulus |
| *output* | An n x J array that the function stores the result in. |

**Returns**

0

This function follows the method outlined in the paper "Single-trial normalization for event-related spectral decomposition reduces sensitivity to noisy trials".

The function will remove the baseline observed in in the pre stimulus by computing the z score on only the information before the stimulus. The variable *m* is the number of samples before the stimulus was introduced.

All arrays must be pre allocated.

## 5.5 /Users/vinay/Google Drive/Science/Research/Morlets/Plot.cc File Reference

All the functions needed to plot the png.

```
#include "wavelet.h"
#include <pngwriter.h>
#include <float.h>
#include <cmath>
```

### Classes

- struct COLOUR

  *the RGB representation of every pixel*
- struct RANGE

  *The lower and upper bound of the data matrix.*

### Macros

- #define PLOT_OY 200

  *The amount of vertical black space in the plot.*
- #define PLOT_OX 200

  *The amount of horizontal black space in the plot.*

### Functions

- RANGE GetRange (double ∗array, int size)
- void CalculateLog (double ∗array, int size)
- COLOUR GetColour (double v, RANGE data_range)
- double Max (double ∗array, int size)

  *A function that finds the Maximum of a given array.*
- double Min (double ∗array, int size)

  *A function that finds the minimum of a given array.*
- void **Plot** (double ∗data, double ∗periods, int num_x, int num_y)

## 5.5.1 Detailed Description

All the functions needed to plot the png.

## 5.5.2 Function Documentation

### 5.5.2.1 void CalculateLog ( double ∗ *array,* int *size* )

**Parameters**

| | |
|---|---|
| *array* | The array that needs to be computed |
| *size* | The size of the contiguous block of memory |

This function will iterate through every element in the array and compute the logarithm. This will override the array.

### 5.5.2.2 COLOUR GetColour ( double *v,* RANGE *data_range* )

**Parameters**

| | |
|---|---|
| *v* | the value of the pixel to be plotted |
| *data_range* | The range of the given data |

**Returns**

pixel_colour The colour of the pixel that will be plotted

This function takes a double and maps to a colour map. High values are closer to the red colour spectrum, and low values are mapped to the blue colour spectrum.

### 5.5.2.3 RANGE GetRange ( double ∗ *array,* int *size* )

**Parameters**

| | |
|---|---|
| *array* | The data array that will be plotted |
| *size* | The total size of the contiguous memory |

**Returns**

RANGE The maximum and minimum of the data array.

This function will iterate through the entire function and returns the maximum and minimum of the entire array.

#### 5.5.2.4 double Max ( double ∗ *array*, int *size* )

A function that finds the Maximum of a given array.

**Parameters**

| | |
|---|---|
| *array* | The array to be analyzed |
| *size* | The size of the array |

#### 5.5.2.5 double Min ( double ∗ *array*, int *size* )

A function that finds the minimum of a given array.

**Parameters**

| | |
|---|---|
| *array* | The array to be analyzed |
| *size* | The size of the array |

## 5.6 /Users/vinay/Google Drive/Science/Research/Morlets/wavelet.cc File Reference

This file contains all of the functions that support the ERSP and CWT functions.

```
#include "wavelet.h"
#include <omp.h>
```

### Macros

- #define **TEST** 0.00001

### Functions

- int Wavelet (double ∗raw_data, double ∗scales, double sampling_frequency, int n, int J, double ∗result)

  *A function that computes the Continuous Wavelet Transform for the data given in raw_data.*
- int CalculatePaddingSize (const int array_size, const int pad_flag)

  *Calculates the size that the padded array should be.*
- double ∗ GenerateScales (const double minimum_frequency, const double maximum_frequency, const double s_0)

  *This function generates the scales that will be used in the Continuous Wavelet Transform.*
- double ∗ IdentifyFrequencies (double ∗scales, int count)

  *Compute the corrosponding frequency to scales used.*
- double CompleteFourierMorlet (const double w, const double scale)

*Computes the Morlet Wavelet in the frequency domain.*
- void TestCases (double ∗data, const int flag)

  *Generates a suite of test case data for wavelet analysis.*
- int WriteFile (const double ∗data, const double ∗frequency, const int x, const int y, const char ∗filename)

  *A function that writes the Wavelet Results to the disk.*
- int WriteDebug (const double ∗data, const int length, const char ∗filename)

  *A function that writes a 1 - d matrix into a log file.*
- void FillData (double ∗data)

  *Populates the input data array with a 3 sparse sine waves.*
- int ReadFile (double data[ ], char filename[ ])

### 5.6.1 Detailed Description

This file contains all of the functions that support the ERSP and CWT functions.

### 5.6.2 Function Documentation

#### 5.6.2.1 int CalculatePaddingSize ( const int *array_size*, const int *pad_flag* )

Calculates the size that the padded array should be.

**Parameters**

| | |
|---|---|
| *array_size* | The cardinal or size of the signal sample |
| *pad_flag* | The type of padding required: see PAD_FLAG |

**Returns**

paadded_size The cardinal or size that the padded array should be

This function computes the size of the padded array depending on the type of padding specified. It takes the size of the data array, and type of pad, and returns how large the padded array should be.

#### 5.6.2.2 double CompleteFourierMorlet ( const double *w*, const double *scale* )

Computes the Morlet Wavelet in the frequency domain.

**Parameters**

| | |
|---|---|
| *w* | |
| *scale* | |

**Returns**

morlet

This function generates the Morlet Wavelet in the frequency domain normalized by the scale.

The formula computed is

$$\hat{\Psi}_\sigma(\omega) = c_\sigma \pi^{-\frac{1}{4}}(e^{-\frac{1}{2}(\sigma-\omega)^2} - \kappa_\sigma e^{-\frac{1}{2}\omega^2})$$

#### 5.6.2.3 void FillData ( double ∗ *data* )

Populates the input data array with a 3 sparse sine waves.

**Parameters**

| | |
|---|---|
| *data* | A 1 - dimentional block of memory that will be overwritten. |

Similar to TestCases() Sine Wave Sample

Sine Wave Sample

#### 5.6.2.4 double∗ GenerateScales ( const double *minimum_frequency,* const double *maximum_frequency,* const double *s_0* )

This function generates the scales that will be used in the Continuous Wavelet Transform.

**Parameters**

| | |
|---|---|
| *minimum_↩ frequency* | The lowest frequency that must be observed |
| *maximum_↩ frequency* | The higest frequency that must be observed |
| *s_0* | The smallest scale usually it is $2 * \delta t$ |

**Returns**

scales An 1 x n array with the dyadic scales.

This function computes the dyadic scales to be generated to accurately compute the multi resolution analysis of a signal. Given the minimum frequency and the maximum frequency, the function will generate a 1 x n array with the scales necessary scale factors for the Continuous Wavelet Transform

The Scales array will be allocated in this function, so it is wise to deallocate this array after it is used.

#### 5.6.2.5 double∗ IdentifyFrequencies ( double ∗ *scales,* int *count* )

Compute the corrosponding frequency to scales used.

**Parameters**

| | |
|---|---|
| *scales* | A 1 x count array of the scales used |
| *count* | The cardinal of the scales array |

**Returns**

frequency The corrosponding frequency for each scale in the scale array

This function computes the corrosponding frequency for each scale provided in the scales array.

It allocates memory and returns the allocated array

It is wise to dealloate this array after use with the free() function.

#### 5.6.2.6 int ReadFile ( double *data[ ],* char *filename[ ]* )

**Parameters**

| | |
|---|---|
| *A* | pre allocated 1 dimentional array |
| *filename* | The name and location of the file to be opened |

**Returns**

array_size The number of elements that was read

This function opens a file, and reads the input assuming that the file is stored with one value at every line.

#### 5.6.2.7 void TestCases ( double * *data*, const int *flag* )

Generates a suite of test case data for wavelet analysis.

**Parameters**

| | |
|---|---|
| *data* | The 1 x n data array to be populated |
| *flag* | The type of test data to be generated |

This function populates the data array with 3 seconds of sample data. The *flag* parameter specifies the type of test data that will be generated

| Flag Type | Output |
|---|---|
| 1 | Impulse at T = 2 seconds |
| 2 | 2 Sine waves at t = 1.5 seconds at FREQ and 2 ∗ FREQ |
| 3 | 2 sine waves at FREQ and 2 ∗ FREQ from t = 0 to 3 s |
| 4 | Single sine wave at t = 1.0 s |
| 5 | sin(x) from t = 0.0 to 3.0 and 2∗sin(x) from t = 1.5 - 2.0 s |
| 6 | sin(x) from t = 0.0 - 3.0 and sin(x - w0) where w0 = 0.005 from t = 1.0 s - 1.5 s |
| 7 | Frequency Sweet from MIN_FREQUENCY to MAX_FREQUENCY |

Table 5.2: TestCases Flags

#### 5.6.2.8 int Wavelet ( double ∗ *raw_data*, double ∗ *scales*, double *sampling_frequency*, int *n*, int *J*, double ∗ *result* )

A function that computes the Continuous Wavelet Transform for the data given in *raw_data*.

**Parameters**

| | |
|---|---|
| *raw_data* | A 1 x n array with the data required |
| *scales* | A 1 x J array with all of the scales for generating the wavelets |
| *sampling_frequency* | The sampling frequency of the given data |
| *n* | The size of the input data |
| *J* | The number of scales that is provided |
| *result* | An n x J array of contiguous memory that stores the result |

This function preforms the Continuous Wavelet Transform using Morlet Wavelets on the data given in raw_data. It stores the result in the result array.

This function only modifies the result array. The arrays must be pre allocated for this function to work.

You can provide the function with scales of your choosing, or one can generate dyadic scales with the GenerateScales() function.

This function is optimized using openmp to allow for multi threading.

#### 5.6.2.9 int WriteDebug ( const double ∗ *data*, const int *length*, const char ∗ *filename* )

A function that writes a 1 - d matrix into a log file.

**Parameters**

| | |
|---|---|
| *data* | A 1 - dimentional data array containing the data to be written |
| *length* | The size of the data array |
| *filename* | THe name of the file to be written |

**Returns**

0 if successful
-1 if unsuccessful

This function writes a 1 - dimentional array to the disk, it's useful when trying to quickly get the results from an array.

#### 5.6.2.10 int WriteFile ( const double ∗ *data*, const double ∗ *frequency*, const int *x*, const int *y*, const char ∗ *filename* )

A function that writes the Wavelet Results to the disk.

**Parameters**

| | |
|---|---|
| *data* | A x x y array with the data that is going to be written |
| *period* | A 1 x y array with the frequencies that were analyzed |
| *x* | The number of samples in the signal |
| *y* | The number of frequencies analyzed |
| *filename* | The name of the file that will be written |

**Returns**

0 if successful
-1 if unsuccessful

This function will write the resultant data computed by Wavelet() and ERSP() into the disk so that it can be graphed by Gnuplot. One can plot the output of this function using the matrix.gplot file.

## 5.7 /Users/vinay/Google Drive/Science/Research/src/FilterTriggers.c File Reference

### Functions

- int FilterTriggers (const int code, const int button, const int numberOfRecords, const long long ∗triggerList, const int ∗readBuffer, int ∗outputBuffer)

  *Filteres the triggers coming in, and finds the specified events.*

### 5.7.1 Function Documentation

#### 5.7.1.1 int FilterTriggers ( const int *code*, const int *button*, const int *numberOfRecords*, const long long ∗ *triggerList*, const int ∗ *readBuffer*, int ∗ *outputBuffer* )

Filteres the triggers coming in, and finds the specified events.

**Parameters**

| | |
|---|---|
| *code* | The code of the trigger list that is needed Possible Inputs: 1, or 2 |
| *button* | The code of the button that is needed Possible Inputs 1, or 2 |
| *triggerList* | The list of all of the possible triggers |
| *readBuffer* | The Status Channel input from the file |
| *outputBuffer* | The buffer that FilterTriggers will populate with the location of the location of the triggers that we're looking for |

**Returns**

counterVariable The number of triggers found.

## 5.8 /Users/vinay/Google Drive/Science/Research/src/FindTriggers.c File Reference

```
#include <stdio.h>
```

## Macros

- #define **MAXIMUM_TRIGGERS** 1000000

## Functions

- long long FindTriggers (const int ∗statusInput, const long long numberOfRecords, long long ∗outputBuffer)
  *This function should take an array input and return the rising and falling edges of the triggers.*

### 5.8.1 Function Documentation

#### 5.8.1.1 long long FindTriggers ( const int ∗ *statusInput,* const long long *numberOfRecords,* long long ∗ *outputBuffer* )

This function should take an array input and return the rising and falling edges of the triggers.

**Parameters**

| | |
|---|---|
| *statusInput* | The Status Channel Input from the BDF or EDF flie. use the edfread_digital_samples |
| *numberOf←Records* | The size of statusInput |
| *outputBuffer* | a 1 x 2 ∗ MAXIMUM_TRIGGERS long long array with the odd entries being the rising edge and the even entries being the falling edges. |

**Returns**

counterVariable The number of triggers that were found.

# Index