
PyEDFlib Documentation

Release 0.1.22

Holger Nahrstaedt

Apr 16, 2021

Contents

1	Description	3
2	Requirements	5
3	Download	7
4	License	9
5	Contents	11
	Index	47

PyEDFlib is a free Open Source wavelet toolbox for reading / writing *EDF/EDF+/BDF* files.

```
import pyedflib
import numpy as np

file_name = pyedflib.data.get_generator_filename()
f = pyedflib.EdfReader(file_name)
n = f.signals_in_file
signal_labels = f.getSignalLabels()
sigbufs = np.zeros((n, f.getNSamples()[0]))
for i in np.arange(n):
    sigbufs[i, :] = f.readSignal(i)
```


CHAPTER 1

Description

`PyEDFlib` is a [Python](#) library to read/write *EDF/EDF+/BDF* files based on `EDFlib`.

EDF stands for [European Data Format](#), a data format for EEG data, first published in 1992. In 2003, an improved version of the file protocol named *EDF+* has been published.

The definition of the *EDF/EDF+* format can be found under edfplus.info.

The *EDF/EDF+* format saves all data with 16 Bit. A version of the format which saves all data with 24 Bit, called *BDF*, was introduced by the company [BioSemi](#).

The `PyEDFlib` [Python](#) toolbox is a fork of the `python-edf` toolbox from [Christopher Lee-Messer](#). and uses the `EDFlib` from Teunis van Beelen.

CHAPTER 2

Requirements

PyEDFlib requires:

- Python ≥ 3.5
- Numpy $\geq 1.9.1$

CHAPTER 3

Download

The most recent *development* version can be found on GitHub at <https://github.com/holgern/pyedflib>.

The latest release, including source and binary package for Windows, is available for download from the [Python Package Index](#) or on the [Releases Page](#).

CHAPTER 4

License

This code is licensed under the same BSD-style license that Teunis released [edflib](#) under and with the same disclaimer.

5.1 API Reference

5.1.1 EDF/BDF file writer `edfwriter`

class `pyedflib.EdfWriter` (*file_name*, *n_channels*, *file_type=1*)
 Bases: `object`

Methods

<code>blockWritePhysicalSamples(self, data)</code>	Writes physical samples (uV, mA, Ohm) must be filled with samples from all signals where each signal has <i>n</i> samples which is the samplefrequency of the signal.
<code>close(self)</code>	Closes the file.
<code>setAdmincode(self, admincode)</code>	Sets the admincode.
<code>setBirthdate(self, birthdate)</code>	Sets the birthdate.
<code>setDatarecordDuration(self, duration)</code>	Sets the datarecord duration.
<code>setDigitalMaximum(self, edfsignal, ...)</code>	Sets the maximum digital value of signal <i>edfsignal</i> .
<code>setDigitalMinimum(self, edfsignal, ...)</code>	Sets the minimum digital value of signal <i>edfsignal</i> .
<code>setEquipment(self, equipment)</code>	Sets the name of the param equipment used during the aquisition.
<code>setGender(self, gender)</code>	Sets the gender.
<code>setHeader(self, fileHeader)</code>	Sets the file header
<code>setLabel(self, edfsignal, label)</code>	Sets the label (name) of signal <i>edfsignal</i> ("FP1", "SaO2", etc.).
<code>setPatientAdditional(self, patient_additional)</code>	Sets the additional patientinfo to <i>patient_additional</i> .
<code>setPatientCode(self, patient_code)</code>	Sets the patientcode to <i>patient_code</i> .

Continued on next page

Table 1 – continued from previous page

<code>setPatientName(self, patient_name)</code>	Sets the patientname to <i>patient_name</i> .
<code>setPhysicalDimension(self, edfsignal, ...)</code>	Sets the physical dimension of signal edfsignal (“uV”, “BPM”, “mA”, “Degr.”, etc.)
<code>setPhysicalMaximum(self, edfsignal, ...)</code>	Sets the physical_maximum of signal edfsignal.
<code>setPhysicalMinimum(self, edfsignal, ...)</code>	Sets the physical_minimum of signal edfsignal.
<code>setPrefilter(self, edfsignal, prefilter)</code>	Sets the prefilter of signal edfsignal (“HP:0.1Hz”, “LP:75Hz N:50Hz”, etc.)
<code>setRecordingAdditional(self, ...)</code>	Sets the additional recordinginfo
<code>setSamplefrequency(self, edfsignal, ...)</code>	Sets the samplefrequency of signal edfsignal.
<code>setSignalHeader(self, edfsignal, channel_info)</code>	Sets the parameter for signal edfsignal.
<code>setSignalHeaders(self, signalHeaders)</code>	Sets the parameter for all signals
<code>setStartdatetime(self, recording_start_time)</code>	Sets the recording start Time
<code>setTechnician(self, technician)</code>	Sets the technicians name to <i>technician</i> .
<code>setTransducer(self, edfsignal, transducer)</code>	Sets the transducer of signal edfsignal
<code>set_number_of_annotation_signals(self, ...)</code>	Sets the number of annotation signals.
<code>update_header(self)</code>	Updates header to edffile struct
<code>writeAnnotation(self, onset_in_seconds, ...)</code>	Writes an annotation/event to the file
<code>writePhysicalSamples(self, data)</code>	Writes n physical samples (uV, mA, Ohm) belonging to one signal where n is the samplefrequency of the signal.
<code>writeSamples(self, data_list[, digital])</code>	Writes physical samples (uV, mA, Ohm) from data belonging to all signals The physical samples will be converted to digital samples using the values of physical maximum, physical minimum, digital maximum and digital minimum.

blockWriteDigitalSamples	
blockWriteDigitalShortSamples	
writeDigitalSamples	
writeDigitalShortSamples	

blockWriteDigitalSamples (*self*, *data*)

blockWriteDigitalShortSamples (*self*, *data*)

blockWritePhysicalSamples (*self*, *data*)

Writes physical samples (uV, mA, Ohm) must be filled with samples from all signals where each signal has n samples which is the samplefrequency of the signal.

data_vec belonging to one signal. The size must be the samplefrequency of the signal.

Notes

buf must be filled with samples from all signals, starting with signal 0, 1, 2, etc. one block equals one second The physical samples will be converted to digital samples using the values of physical maximum, physical minimum, digital maximum and digital minimum The number of samples written is equal to the sum of the samplefrequencies of all signals Size of *buf* should be equal to or bigger than `sizeof(double)` multiplied by the sum of the samplefrequencies of all signals Returns 0 on success, otherwise -1

All parameters must be already written into the bdf/edf-file.

close (*self*)

Closes the file.

setAdmincode (*self*, *admincode*)

Sets the admincode.

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

Parameters

admincode [str] admincode which is written into the header

setBirthdate (*self*, *birthdate*)

Sets the birthdate.

Parameters

birthdate: date object from datetime

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

Examples

```
>>> import pyedflib
>>> from datetime import datetime, date
>>> f = pyedflib.EdfWriter('test.bdf', 1, file_type=pyedflib.FILETYPE_BDFPLUS)
>>> f.setBirthdate(date(1951, 8, 2))
>>> f.close()
```

setDatarecordDuration (*self*, *duration*)

Sets the datarecord duration. The default value is 100000 which is 1 second. ATTENTION: the argument “duration” is expressed in units of 10 microseconds! So, if you want to set the datarecord duration to 0.1 second, you must give the argument “duration” a value of “10000”. This function is optional, normally you don’t need to change the default value. The datarecord duration must be in the range 0.001 to 60 seconds. Returns 0 on success, otherwise -1.

Parameters

duration [integer] Sets the datarecord duration in units of 10 microseconds

Notes

This function is NOT REQUIRED but can be called after opening a file in writemode and before the first sample write action. This function can be used when you want to use a samplerate which is not an integer. For example, if you want to use a samplerate of 0.5 Hz, set the samplefrequency to 5 Hz and the datarecord duration to 10 seconds. Do not use this function, except when absolutely necessary!

setDigitalMaximum (*self*, *edfsignal*, *digital_maximum*)

Sets the maximum digital value of signal edfsignal. Usually, the value 32767 is used for EDF+ and 8388607 for BDF+.

Parameters

edfsignal [int] signal number

digital_maximum [int] Sets the maximum digital value

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

setDigitalMinimum (*self*, *edfsignal*, *digital_minimum*)

Sets the minimum digital value of signal *edfsignal*. Usually, the value -32768 is used for EDF+ and -8388608 for BDF+. Usually this will be $-(\text{digital_maximum} + 1)$.

Parameters

edfsignal [int] signal number

digital_minimum [int] Sets the minimum digital value

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

setEquipment (*self*, *equipment*)

Sets the name of the param equipment used during the acquisition. This function is optional and can be called only after opening a file in writemode and before the first sample write action.

Parameters

equipment [str] Describes the measurement equipment

setGender (*self*, *gender*)

Sets the gender. This function is optional and can be called only after opening a file in writemode and before the first sample write action.

Parameters

gender [int] 1 is male, 0 is female

setHeader (*self*, *fileHeader*)

Sets the file header

setLabel (*self*, *edfsignal*, *label*)

Sets the label (name) of signal *edfsignal* ("FP1", "SaO2", etc.).

Parameters

edfsignal [int] signal number on which the label should be changed

label [str] signal label

Notes

This function is recommended for every signal and can be called only after opening a file in writemode and before the first sample write action.

setPatientAdditional (*self*, *patient_additional*)

Sets the additional patientinfo to *patient_additional*.

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

setPatientCode (*self*, *patient_code*)
Sets the patientcode to *patient_code*.

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

setPatientName (*self*, *patient_name*)
Sets the patientname to *patient_name*.

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

setPhysicalDimension (*self*, *edfsignal*, *physical_dimension*)
Sets the physical dimension of signal *edfsignal* (“uV”, “BPM”, “mA”, “Degr.”, etc.)

Parameters

- **edfsignal** – int
- **physical_dimension** – str

Notes

This function is recommended for every signal and can be called only after opening a file in writemode and before the first sample write action.

setPhysicalMaximum (*self*, *edfsignal*, *physical_maximum*)
Sets the *physical_maximum* of signal *edfsignal*.

Parameters

- edfsignal: int** signal number
- physical_maximum: float** Sets the physical maximum

Notes

This function is required for every signal and can be called only after opening a file in writemode and before the first sample write action.

setPhysicalMinimum (*self*, *edfsignal*, *physical_minimum*)
Sets the *physical_minimum* of signal *edfsignal*.

Parameters

- edfsignal: int** signal number
- physical_minimum: float** Sets the physical minimum

Notes

This function is required for every signal and can be called only after opening a file in writemode and before the first sample write action.

setPrefilter (*self*, *edfsignal*, *prefilter*)

Sets the prefilter of signal edfsignal (“HP:0.1Hz”, “LP:75Hz N:50Hz”, etc.)

Parameters

- **edfsignal** – int
- **prefilter** – str

Notes

This function is optional for every signal and can be called only after opening a file in writemode and before the first sample write action.

setRecordingAdditional (*self*, *recording_additional*)

Sets the additional recordinginfo

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

setSamplefrequency (*self*, *edfsignal*, *samplefrequency*)

Sets the samplefrequency of signal edfsignal.

Notes

This function is required for every signal and can be called only after opening a file in writemode and before the first sample write action.

setSignalHeader (*self*, *edfsignal*, *channel_info*)

Sets the parameter for signal edfsignal.

channel_info should be a dict with these values:

‘label’ : channel label (string, <= 16 characters, must be unique) ‘dimension’ : physical dimension (e.g., mV) (string, <= 8 characters) ‘sample_rate’ : sample frequency in hertz (int) ‘physical_max’ : maximum physical value (float) ‘physical_min’ : minimum physical value (float) ‘digital_max’ : maximum digital value (int, $-2^{15} \leq x < 2^{15}$) ‘digital_min’ : minimum digital value (int, $-2^{15} \leq x < 2^{15}$)

setSignalHeaders (*self*, *signalHeaders*)

Sets the parameter for all signals

Parameters

signalHeaders [array_like]

containing dict with

- ‘label’ [str] channel label (string, <= 16 characters, must be unique)
- ‘dimension’ [str] physical dimension (e.g., mV) (string, <= 8 characters)
- ‘sample_rate’ [int] sample frequency in hertz

‘physical_max’ [float] maximum physical value

‘physical_min’ [float] minimum physical value

‘digital_max’ [int] maximum digital value ($-2^{15} \leq x < 2^{15}$)

‘digital_min’ [int] minimum digital value ($-2^{15} \leq x < 2^{15}$)

setStartdatetime (*self, recording_start_time*)

Sets the recording start Time

Parameters

recording_start_time: datetime object Sets the recording start Time

setTechnician (*self, technician*)

Sets the technicians name to *technician*.

Notes

This function is optional and can be called only after opening a file in writemode and before the first sample write action.

setTransducer (*self, edfsignal, transducer*)

Sets the transducer of signal edfsignal

Parameters

- **edfsignal** – int
- **transducer** – str

Notes

This function is optional for every signal and can be called only after opening a file in writemode and before the first sample write action.

set_number_of_annotation_signals (*self, number_of_annotations*)

Sets the number of annotation signals. The default value is 1 This function is optional and can be called only after opening a file in writemode and before the first sample write action Normally you don't need to change the default value. Only when the number of annotations you want to write is more than the number of seconds of the duration of the recording, you can use this function to increase the storage space for annotations Minimum is 1, maximum is 64

Parameters

number_of_annotations [integer] Sets the number of annotation signals

update_header (*self*)

Updates header to edffile struct

writeAnnotation (*self, onset_in_seconds, duration_in_seconds, description, str_format='utf-8'*)

Writes an annotation/event to the file

writeDigitalSamples (*self, data*)

writeDigitalShortSamples (*self, data*)

writePhysicalSamples (*self, data*)

Writes n physical samples (uV, mA, Ohm) belonging to one signal where n is the samplefrequency of the signal.

data_vec belonging to one signal. The size must be the samplefrequency of the signal.

Notes

Writes n physical samples (uV, mA, Ohm) from `data_vec` belonging to one signal where n is the sample-frequency of the signal. The physical samples will be converted to digital samples using the values of physical maximum, physical minimum, digital maximum and digital minimum. The number of samples written is equal to the samplefrequency of the signal. Call this function for every signal in the file. The order is important! When there are 4 signals in the file, the order of calling this function must be: signal 0, signal 1, signal 2, signal 3, signal 0, signal 1, signal 2, etc.

All parameters must be already written into the bdf/edf-file.

writeSamples (*self*, *data_list*, *digital=False*)

Writes physical samples (uV, mA, Ohm) from data belonging to all signals The physical samples will be converted to digital samples using the values of physical maximum, physical minimum, digital maximum and digital minimum. if the samplefrequency of all signals are equal, then the data could be saved into a matrix with the size (N,signals) If the samplefrequency is different, then `sample_freq` is a vector containing all the different samplefrequencys. The data is saved as list. Each list entry contains a vector with the data of one signal.

If `digital` is `True`, digital signals (as directly from the ADC) will be expected. (e.g. int16 from 0 to 2048)

All parameters must be already written into the bdf/edf-file.

This section describes functions used to perform reading of a EDF/BDF file.

5.1.2 EDF/BDF file reader `edfreader`

class `pyedflib.EdfReader`

Bases: `pyedflib._extensions._pyedflib.CyEdfReader`

This provides a simple interface to read EDF, EDF+, BDF and BDF+ files.

Attributes

admincode

annotations_in_file

birthdate

datarecord_duration datarecord duration in seconds (as a double)

datarecords_in_file number of data records

equipment

file_duration file duration in seconds

filetype

gender

handle edflib internal int handle

patient patient name?

patient_additional

patientcode

patientname

recording_additional

signals_in_file
startdate_day
startdate_month
startdate_year
starttime_hour
starttime_minute
starttime_second
starttime_subsecond
technician

Methods

<i>close(self)</i>	Closes the file handler
<i>file_info_long(self)</i>	Returns information about the opened EDF/BDF file
<i>getAdmincode(self)</i>	Returns the Admincode.
<i>getBirthdate(self[, string])</i>	Returns the birthdate as string object
<i>getDigitalMaximum(self[, chn])</i>	Returns the maximum digital value of signal edfsignal.
<i>getDigitalMinimum(self[, chn])</i>	Returns the minimum digital value of signal edfsignal.
<i>getEquipment(self)</i>	Returns the used Equipment.
<i>getFileDuration(self)</i>	Returns the duration of the file in seconds.
<i>getGender(self)</i>	Returns the Gender of the patient.
<i>getHeader(self)</i>	Returns the file header as dict
<i>getLabel(self, chn)</i>	Returns the label (name) of signal chn ("FP1", "SaO2", etc.).
<i>getPatientAdditional(self)</i>	Returns the additional patientinfo.
<i>getPatientCode(self)</i>	Returns the patientcode
<i>getPatientName(self)</i>	Returns the patientname
<i>getPhysicalDimension(self, chn)</i>	Returns the physical dimension of signal edfsignal ("uV", "BPM", "mA", "Degr.", etc.).
<i>getPhysicalMaximum(self[, chn])</i>	Returns the maximum physical value of signal edfsignal.
<i>getPhysicalMinimum(self[, chn])</i>	Returns the minimum physical value of signal edfsignal.
<i>getPrefilter(self, chn)</i>	Returns the prefilter of signal chn ("HP:0.1Hz", "LP:75Hz N:50Hz", etc.).
<i>getRecordingAdditional(self)</i>	Returns the additional recordinginfo
<i>getSampleFrequencies(self)</i>	Returns samplefrequencies of all signals.
<i>getSampleFrequency(self, chn)</i>	Returns the samplefrequency of signal edfsignal.
<i>getSignalHeader(self, chn)</i>	Returns the header of one signal as dicts
<i>getSignalHeaders(self)</i>	Returns the header of all signals as array of dicts
<i>getSignalLabels(self)</i>	Returns all labels (name) ("FP1", "SaO2", etc.).
<i>getStartdatetime(self)</i>	Returns the date and starttime as datetime object
<i>getTechnician(self)</i>	Returns the technicians name
<i>getTransducer(self, chn)</i>	Returns the transducer of signal chn ("AgAgCl cup electrodes", etc.).

Continued on next page

Table 2 – continued from previous page

<code>make_buffer()</code>	utility function to make a buffer that can hold a single datarecord.
<code>open(file_name, annotations_mode, ...)</code>	
<code>readAnnotations(self)</code>	Annotations from a edf-file
<code>readSignal(self, chn[, start, n, digital])</code>	Returns the physical data of signal chn.
<code>read_digital_signal()</code>	<code>read_digital_signal(self, signalnum, start, n, np.ndarray[np.int32_t, ndim=1] sigbuf read @n number of samples from signal number @signum starting at @start into numpy int32 array @sigbuf sigbuf must be at least n long</code>
<code>readsignal()</code>	<code>read @n number of samples from signal number @signum starting at @start into numpy float64 array @sigbuf sigbuf must be at least n long</code>

check_open_ok	
digital_max	
digital_min	
file_info	
getNSamples	
load_datarecord	
physical_dimension	
physical_max	
physical_min	
prefilter	
read_annotation	
samplefrequency	
samples_in_datarecord	
samples_in_file	
signal_label	
transducer	

close (*self*)

Closes the file handler

file_info (*self*)

file_info_long (*self*)

Returns information about the opened EDF/BDF file

getAdmincode (*self*)

Returns the Admincode.

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getAdmincode() == ' '
True
>>> f.close()
```


getBirthdate (*self*, *string=True*)

Returns the birthdate as string object

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getBirthdate()=='30 jun 1969'
True
>>> f.close()
```

getDigitalMaximum (*self*, *chn=None*)

Returns the maximum digital value of signal edfsignal.

Parameters

chn [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getDigitalMaximum(0)
32767
>>> f.close()
```

getDigitalMinimum (*self*, *chn=None*)

Returns the minimum digital value of signal edfsignal.

Parameters

chn [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getDigitalMinimum(0)
-32768
>>> f.close()
```

getEquipment (*self*)

Returns the used Equipment.

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getEquipment()=='test generator'
True
>>> f.close()
```

getFileDuration (*self*)

Returns the duration of the file in seconds.

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getFileDuration()==600
True
>>> f.close()
```

getGender (*self*)

Returns the Gender of the patient.

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getGender()==' '
True
>>> f.close()
```

getHeader (*self*)

Returns the file header as dict

Parameters

None

getLabel (*self*, *chn*)

Returns the label (name) of signal chn (“FP1”, “SaO2”, etc.).

Parameters

chn [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getLabel(0)=='squarewave'
True
>>> f.close()
```

getNSamples (*self*)

getPatientAdditional (*self*)

Returns the additional patientinfo.

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getPatientAdditional()==' '
True
>>> f.close()
```

getPatientCode (*self*)

Returns the patientcode

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getPatientCode()==' '
True
>>> f.close()
```

getPatientName (*self*)

Returns the patientname

Parameters

None

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getPatientName()=='X'
True
>>> f.close()
```

getPhysicalDimension (*self*, *chn*)

Returns the physical dimension of signal edfsignal (“uV”, “BPM”, “mA”, “Degr.”, etc.)

Parameters

chn [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getPhysicalDimension(0)=='uV'
True
>>> f.close()
```

getPhysicalMaximum (*self*, *chn=None*)

Returns the maximum physical value of signal edfsignal.

Parameters

chn [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getPhysicalMaximum(0)==1000.0
True
>>> f.close()
```

getPhysicalMinimum (*self*, *chn=None*)

Returns the minimum physical value of signal edfsignal.

Parameters

chn [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getPhysicalMinimum(0)==-1000.0
True
>>> f.close()
```

getPrefilter (*self*, *chn*)

Returns the prefilter of signal *chn* (“HP:0.1Hz”, “LP:75Hz N:50Hz”, etc.)

Parameters

chn [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getPrefilter(0)==' '
```

(continues on next page)

(continued from previous page)

```
True
>>> f.close()
```

getRecordingAdditional (*self*)

Returns the additional recordinginfo

Parameters**None****Examples**

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getRecordingAdditional()==' '
True
>>> f.close()
```

getSampleFrequencies (*self*)

Returns samplefrequencies of all signals.

Parameters**None****Examples**

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> all(f.getSampleFrequencies()==200.0)
True
>>> f.close()
```

getSampleFrequency (*self*, *chn*)

Returns the samplefrequency of signal edfsignal.

Parameters**chn** [int] channel number**Examples**

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getSampleFrequency(0)==200.0
True
>>> f.close()
```

getSignalHeader (*self*, *chn*)

Returns the header of one signal as dicts

Parameters**None**

getSignalHeaders (*self*)

Returns the header of all signals as array of dicts

Parameters**None****getSignalLabels** (*self*)

Returns all labels (name) (“FP1”, “SaO2”, etc.).

Parameters**None**

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getSignalLabels() == ['squarewave', 'ramp', 'pulse', 'noise', 'sine 1 Hz',
→ 'sine 8 Hz', 'sine 8.1777 Hz', 'sine 8.5 Hz', 'sine 15 Hz', 'sine 17 Hz',
→ 'sine 50 Hz']
True
>>> f.close()
```

getStartdatetime (*self*)

Returns the date and starttime as datetime object

Parameters**None**

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getStartdatetime()
datetime.datetime(2011, 4, 4, 12, 57, 2)
>>> f.close()
```

getTechnician (*self*)

Returns the technicians name

Parameters**None****.. code-block:: python**

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getTechnician() == ''
True
>>> f.close()
```

getTransducer (*self*, *chn*)Returns the transducer of signal *chn* (“AgAgCl cup electrodes”, etc.).**Parameters****chn** [int] channel number

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> f.getTransducer(0) == ' '
True
>>> f.close()
```

readAnnotations (*self*)

Annotations from a edf-file

Parameters

None

readSignal (*self*, *chn*, *start=0*, *n=None*, *digital=False*)

Returns the physical data of signal *chn*. When *start* and *n* is set, a subset is returned

Parameters

chn [int] channel number

start [int] start pointer (default is 0)

n [int] length of data to read (default is None, by which the complete data of the channel are returned)

digital: bool will return the signal in original digital values instead of physical values

Examples

```
>>> import pyedflib
>>> f = pyedflib.data.test_generator()
>>> x = f.readSignal(0,0,1000)
>>> int(x.shape[0])
1000
>>> x2 = f.readSignal(0)
>>> int(x2.shape[0])
120000
>>> f.close()
```

5.1.3 EDF/BDF highlevel functions

`pyedflib.highlevel.change_polarity` (*edf_file*, *channels*, *new_file=None*, *verify=True*, *verbose=False*)

Change polarity of certain channels

Parameters

edf_file [str] from which file to change polarity.

channels [list of int] the indices of the channels.

new_file [str, optional] where to save the edf with inverted channels. The default is None.

verify [bool, optional] whether to verify the two edfs for similarity. The default is True.

verbose [str, optional] print progress or not. The default is True.

Returns

bool True if success.

`pyedflib.highlevel.rename_channels(edf_file, mapping, new_file=None, verbose=False)`
A convenience function to rename channels in an EDF file.

Parameters

edf_file [str] an string pointing to an edf file.

mapping [dict] a dictionary with channel mappings as key:value. eg: {'M1-O2': 'A1-O2'}

new_file [str, optional] the new filename. If None will be edf_file + '_renamed' The default is None.

verbose [bool, optional] print progress or not. The default is False.

Returns

bool True if successful, False if failed.

`pyedflib.highlevel.anonymize_edf(edf_file, new_file=None, to_remove=['patientname', 'birthdate'], new_values=['xxx', ''], verify=False, verbose=False)`
Anonymize an EDF file by replacing values of header fields.

This function can be used to overwrite all header information that is patient specific, for example birthdate and patientname. All header fields can be overwritten this way (i.e., all `header.keys()` given `_, _, header = read_edf(edf_file, digital=True)`).

Parameters

edf_file [str] Filename of an EDF/BDF.

new_file [str | None] The filename of the anonymized file. If None, the input filename appended with '_anonymized' is used. Defaults to None.

to_remove [list of str] List of attributes to overwrite in the *edf_file*. Defaults to ['patientname', 'birthdate'].

new_values [list of str] List of values used for overwriting the attributes specified in *to_remove*. Each item in *to_remove* must have a corresponding item in *new_values*. Defaults to ['xxx', ''].

verify [bool] Compare *edf_file* and *new_file* for equality (i.e., double check that values are same). Defaults to False

verbose [bool, optional] print progress or not. The default is False.

Returns

bool True if successful, or if *verify* is False. Raises an error otherwise.

`pyedflib.highlevel.drop_channels(edf_source, edf_target=None, to_keep=None, to_drop=None, verbose=False)`
Remove channels from an edf file. Save the file. For safety reasons, no source files can be overwritten.

Parameters

edf_source [str] The source edf file from which to drop channels.

edf_target [str, optional] Where to save the file. If None, will be *edf_source* + 'dropped.edf'. The default is None.

to_keep [list, optional] A list of channel names or indices that will be kept. Strings will always be interpreted as channel names. 'to_keep' will overwrite any droppings proposed by to_drop. The default is None.

to_drop [list, optional] A list of channel names/indices that should be dropped. Strings will be interpreted as channel names. The default is None.

verbose [bool, optional] print progress or not. The default is False.

Returns

edf_target [str] the target filename with the dropped channels.

`pyedflib.highlevel.compare_edf(edf_file1, edf_file2, verbose=False)`

Loads two edf files and checks whether the values contained in them are the same. Does not check the header or annotations data.

Mainly to verify that other options (eg anonymization) produce the same EDF file.

Parameters

edf_file1 [str] edf file 1 to compare.

edf_file2 [str] edf file 2 to compare.

verbose [bool, optional] print progress or not. The default is False.

Returns

bool True if signals are equal, else raises error.

`pyedflib.highlevel.read_edf_header(edf_file, read_annotations=True)`

Reads the header and signal headers of an EDF file and it's annotations

Parameters

edf_file [str] EDF/BDF file to read.

Returns

summary [dict] header of the edf file as dictionary.

`pyedflib.highlevel.write_edf_quick(edf_file, signals, sfreq, digital=False)`

wrapper for write_pyedf without creating headers. Use this if you don't care about headers or channel names and just want to dump some signals with the same sampling freq. to an edf

Parameters

edf_file [str] where to store the data/edf.

signals [np.ndarray] The signals you want to store as numpy array.

sfreq [int] the sampling frequency of the signals.

digital [bool, optional] if the data is present digitally (int) or as mV/uV. The default is False.

Returns

bool True if successful, else False or raise Error.

`pyedflib.highlevel.write_edf(edf_file, signals, signal_headers, header=None, digital=False, file_type=-1, block_size=1)`

Write signals to an edf_file. Header can be generated on the fly with generic values. EDF+/BDF+ is selected based on the filename extension, but can be overwritten by setting filetype to pyedflib.FILETYPE_XXX

Parameters

edf_file [np.ndarray or list] where to save the EDF file

signals [list] The signals as a list of arrays or a ndarray.

signal_headers [list of dict] a list with one signal header(dict) for each signal. See `pyedflib.EdfWriter.setSignalHeader`.

header [dict] a main header (dict) for the EDF file, see `pyedflib.EdfWriter.setHeader` for details. If no header present, will create an empty header

digital [bool, optional] whether the signals are in digital format (ADC). The default is False.

filetype: int, optional choose filetype for saving. EDF = 0, EDF+ = 1, BDF = 2, BDF+ = 3, automatic from extension = -1

block_size [int] set the block size for writing. Should be divisor of signal length in seconds. Higher values mean faster writing speed, but if it is not a divisor of the signal duration, it will append zeros. Can be any value between 1=><=60, -1 will auto-infer the fastest value.

Returns

bool True if successful, False if failed.

`pyedflib.highlevel.read_edf(edf_file, ch_nrs=None, ch_names=None, digital=False, verbose=False)`

Convenience function for reading EDF+/BDF data with `pyedflib`.

Will load the edf and return the signals, the headers of the signals and the header of the EDF. If all signals have the same sample frequency will return a numpy array, else a list with the individual signals

Parameters

edf_file [str] link to an edf file.

ch_nrs [list of int, optional] The indices of the channels to read. The default is None.

ch_names [list of str, optional] The names of channels to read. The default is None.

digital [bool, optional] will return the signals as digital values (ADC). The default is False.

verbose [bool, optional] Print progress bar while loading or not. The default is False.

Returns

signals [np.ndarray or list] the signals of the chosen channels contained in the EDF.

signal_headers [list] one signal header for each channel in the EDF.

header [dict] the main header of the EDF file containing meta information.

`pyedflib.highlevel.make_signal_headers(list_of_labels, dimension='uV', sample_rate=256, physical_min=-200.0, physical_max=200.0, digital_min=-32768, digital_max=32767, transducer="", prefilter="")`

A function that creates signal headers for a given list of channel labels. This can only be used if each channel has the same sampling frequency

Parameters

list_of_labels [list of str] A list with labels for each channel.

dimension [str, optional] dimension, eg mV. The default is 'uV'.

sample_rate [int, optional] sampling frequency. The default is 256.

physical_min [float, optional] minimum value in dimension. The default is -200.

physical_max [float, optional] maximum value in dimension. The default is 200.

digital_min [int, optional] digital minimum of the ADC. The default is -32768.

digital_max [int, optional] digital maximum of the ADC. The default is 32767.

transducer [str, optional] electrode type that was used. The default is ‘’.

prefiler [str, optional] filtering and sampling method. The default is ‘’.

Returns

signal_headers [list of dict] returns n signal headers as a list to save several signal headers.

```
pyedflib.highlevel.make_signal_header (label,          dimension='uV',      sample_rate=256,
                                       physical_min=-200,    physical_max=200,
                                       digital_min=-32768,   digital_max=32767,   trans-
                                       ducer='', prefiler='')
```

A convenience function that creates a signal header for a given signal. This can be used to create a list of signal headers that is used by pyedflib to create an edf. With this, different sampling frequencies can be indicated.

Parameters

label [str] the name of the channel.

dimension [str, optional] dimension, eg mV. The default is ‘uV’.

sample_rate [int, optional] sampling frequency. The default is 256.

physical_min [float, optional] minimum value in dimension. The default is -200.

physical_max [float, optional] maximum value in dimension. The default is 200.

digital_min [int, optional] digital minimum of the ADC. The default is -32768.

digital_max [int, optional] digital maximum of the ADC. The default is 32767.

transducer [str, optional] electrode type that was used. The default is ‘’.

prefiler [str, optional] filtering and sampling method. The default is ‘’.

Returns

signal_header [dict] a signal header that can be used to save a channel to an EDF.

```
pyedflib.highlevel.make_header (technician="",    recording_additional="",    patientname="",
                                patient_additional="", patientcode="",    equipment="",    admin-
                                code="", gender="", startdate=None, birthdate="")
```

A convenience function to create an EDF header (a dictionary) that can be used by pyedflib to update the main header of the EDF

Parameters

technician [str, optional] name of the technician. The default is ‘’.

recording_additional [str, optional] comments etc. The default is ‘’.

patientname [str, optional] the name of the patient. The default is ‘’.

patient_additional [TYPE, optional] more info about the patient. The default is ‘’.

patientcode [str, optional] alphanumeric code. The default is ‘’.

equipment [str, optional] which system was used. The default is ‘’.

admincode [str, optional] code of the admin. The default is ‘’.

gender [str, optional] gender of patient. The default is ‘’.

startdate [datetime.datetime, optional] startdate of recording. The default is None.

birthdate [str/datetime.datetime, optional] date of birth of the patient. The default is ‘’.

Returns

header [dict] a dictionary with the values given filled in.

`pyedflib.highlevel.phys2dig` (*signal*, *dmin*, *dmax*, *pmin*, *pmax*)
converts physical values to digital values

Parameters

signal [np.ndarray or int] A numpy array with int values (digital values) or an int.

dmin [int] digital minimum value of the edf file (eg -2048).

dmax [int] digital maximum value of the edf file (eg 2048).

pmin [float] physical maximum value of the edf file (eg -200.0).

pmax [float] physical maximum value of the edf file (eg 200.0).

Returns

digital [np.ndarray or int] converted digital values

`pyedflib.highlevel.dig2phys` (*signal*, *dmin*, *dmax*, *pmin*, *pmax*)
converts digital edf values to physical values

Parameters

signal [np.ndarray or int] A numpy array with int values (digital values) or an int.

dmin [int] digital minimum value of the edf file (eg -2048).

dmax [int] digital maximum value of the edf file (eg 2048).

pmin [float] physical maximum value of the edf file (eg -200.0).

pmax [float] physical maximum value of the edf file (eg 200.0).

Returns

physical [np.ndarray or float] converted physical values

5.2 Development notes

This section contains information on building and installing PyEDFlib from source code as well as instructions for preparing the build environment on Windows and Linux.

5.2.1 Preparing Windows build environment

To start developing pyedflib code on Windows you will have to install a C compiler and prepare the build environment.

Installing Microsoft Visual C++ Compiler for Python 2.7

Downloading Microsoft Visual C++ Compiler for Python 2.7 from <https://www.microsoft.com/en-us/download/details.aspx?id=44266>.

After installing the Compiler and before compiling the extension you have to configure some environment variables.

For build execute the `util/setenv_win.bat` script in the cmd window:

```
rem Configure the environment for builds.
rem Convince setup.py to use the SDK tools.
set MSSdk=1
set DISTUTILS_USE_SDK=1
```

Next steps

After completing these steps continue with *Installing build dependencies*.

5.2.2 Preparing Linux build environment

There is a good chance that you already have a working build environment. Just skip steps that you don't need to execute.

Installing basic build tools

Note that the example below uses `aptitude` package manager, which is specific to Debian and Ubuntu Linux distributions. Use your favourite package manager to install these packages on your OS.

```
aptitude install build-essential gcc python-dev git-core
```

Next steps

After completing these steps continue with *Installing build dependencies*.

5.2.3 Installing build dependencies

Setting up Python virtual environment

A good practice is to create a separate Python virtual environment for each project. If you don't have `virtualenv` yet, install and activate it using:

```
curl -O https://raw.githubusercontent.com/pypa/virtualenv/master/virtualenv.py
python virtualenv.py <name_of_the_venv>
. <name_of_the_venv>/bin/activate
```

Installing Cython

Use `pip` (<http://pypi.python.org/pypi/pip>) to install Cython:

```
pip install Cython>=0.16
```

Installing numpy

Use `pip` to install `numpy`:

```
pip install numpy
```

Numpy can also be obtained via scientific python distributions such as:

- Anaconda
- Enthought Canopy
- Python(x,y)

Note: You can find binaries for 64-bit Windows on <http://www.lfd.uci.edu/~gohlke/pythonlibs/>.

Installing Sphinx

Sphinx is a documentation tool that converts reStructuredText files into nicely looking html documentation. Install it with:

```
pip install Sphinx
```

numpydoc is used to format the API documentation appropriately. Install it via:

```
pip install numpydoc
```

5.2.4 Building and installing PyEDFlib

Installing from source code

Go to <https://github.com/holgern/pyedflib> GitHub project page, fork and clone the repository or use the upstream repository to get the source code:

```
git clone https://github.com/holgern/pyedflib.git pyedflib
```

Activate your Python virtual environment, go to the cloned source directory and type the following commands to build and install the package:

```
python setup.py build
python setup.py install
```

To verify the installation run the following command:

```
python setup.py test
```

To build docs:

```
cd doc
make html
```

Installing from source code in Windows

Go to <https://github.com/holgern/pyedflib> GitHub project page, fork and clone the repository or use the upstream repository to get the source code:

```
git clone https://github.com/holgern/pyedflib.git pyedflib
```

Install Microsoft Visual C++ Compiler for Python 2.7 from <https://www.microsoft.com/en-us/download/details.aspx?id=44266>

Activate your Python virtual environment, go to the cloned source directory and type the following commands to build and install the package:

```
util\setenv_win.bat
python setup.py build_ext --inplace
python setup.py install --user
```

To verify the installation run the following command:

```
python runtests.py
```

To build docs:

```
cd doc
make html
```

Installing a development version

You can also install directly from the source repository:

```
pip install -e git+https://github.com/holgern/pyedflib.git#egg=pyedflib
```

or:

```
pip install pyedflib==dev
```

Installing a regular release from PyPi

A regular release can be installed with pip or easy_install:

```
pip install pyedflib
```

5.2.5 Testing

Continuous integration with Travis-CI

The project is using [Travis-CI](#) service for continuous integration and testing.

Current build status is:

build **passing**

If you are submitting a patch or pull request please make sure it does not break the build.

Running tests locally

Tests are implemented with [nose](#), so use one of:

```
$ nosetests pyedflib
```

```
>>> pyedflib.test() # doctest: +SKIP
```

Note doctests require [Matplotlib](#) in addition to the usual dependencies.

Running tests with Tox

There's also a config file for running tests with [Tox](#) (`pip install tox`). To for example run tests for Python 2.7 and Python 3.4 use:

```
tox -e py27,py34
```

For more information see the [Tox](#) documentation.

5.2.6 Guidelines for new releases for pyedflib

vX.X.X refers to the release number

Tag the release and trigger bulding of wheels in appvoyer

Change ISRELEASED in setup.py to True and commit.

Appveyor will now build wheels for windows.

Tag the relase with

```
`git tag -s vX.X.X`
```

and push the tag to master.

Clean up source

Remove untraced files with `git clean`

First check which files will be deleted:

```
`git clean -x`
```

Then run without -n:

```
`git clean -x`
```

Create the source distribution files with:

```
`python setup.py sdist --formats=gztar,zip`
```


Upload the release and windows wheels to pypi

Download all wheels from Appveyor and put them into the dist directory

Register all files with

```
`twine register dist\filename_which_should_uploaded.whl`
```

and upload with

```
`twine upload dist\filename_which_should_uploaded.whl`
```

Prepare for continued development

Increment the version number in setup.py and change ISRELEASED to False.

5.2.7 Something not working?

If these instructions are not clear or you need help setting up your development environment, go ahead and open a ticket on [GitHub](#).

5.3 Resources

5.3.1 Code

The [GitHub repository](#) is now the main code repository.

If you are using the Mercurial repository at Bitbucket, please switch to Git/GitHub and follow for development updates.

5.3.2 Questions and bug reports

Use [GitHub Issues](#) to post questions and open tickets.

5.4 PyEDFlib

5.4.1 Release Notes

PyEDFlib 0.1.6 Release Notes

Contents

- *PyEDFlib 0.1.6 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.6*
 - * *Pull requests for v0.1.6*

- Faster writing of BDF/EDF files

- bug fixes

Authors

- Holger Nahrstaedt

Issues closed for v0.1.6

- #5: edf writer code executing really slow

Pull requests for v0.1.6

PyEDFlib 0.1.7 Release Notes

Contents

- *PyEDFlib 0.1.7 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.7*
 - * *Pull requests for v0.1.7*

- Fix Crash on 64 Bit
- Fix strings on python 3
- Fix edf handle
- Fix getSignalHeaders

Authors

- Matthias Klumpp +
- Holger Nahrstaedt
- Jukka Zitting +

A total of 3 people contributed to this release. People with a “+” by their names contributed a patch for the first time. This list of names is automatically generated, and may not be fully complete.

Issues closed for v0.1.7

- #7: Handling of wrong paths
- #8: Wrong and misleading docstring for setDatarecordDuration
- #9: How to close an already opened file?
- #10: Incorrect results from calling read_digital_signal when opening more than one file

Pull requests for v0.1.7

- [#12](#): Fix getSignalHeaders
- [#13](#): Don't crash when writing files on 64bit machines

PyEDFlib 0.1.8 Release Notes

Contents

- *PyEDFlib 0.1.8 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.8*

- Fix proper reading of EDF/BDF files
- Fix writing of EDF/BDF files

Authors

- Holger Nahrstaedt

Issues closed for v0.1.8

- [#11](#): IOError: malloc error
- [#14](#): CHB-MIT (Physionet) error - regular EDF not supported anymore

PyEDFlib 0.1.9 Release Notes

Contents

- *PyEDFlib 0.1.9 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.9*
 - * *Pull requests for v0.1.9*

- Fix proper reading of EDF/BDF files
- Bug fixes
- Improved error messages

Authors

- Zaccharie Ramzi +
- Holger Nahrstaedt

Issues closed for v0.1.9

- #16: Double rounding when writing to file leads to accumulating errors
- #17: Error with chb17b_69 in CHB-MIT (Physionet) Database

Pull requests for v0.1.9

- #15: Essentially corrected a typo in readme

PyEDFlib 0.1.10 Release Notes

Contents

- *PyEDFlib 0.1.10 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.10*

- Add function for access subset of data
- Improved error messages

Authors

- Holger Nahrstaedt

Issues closed for v0.1.10

- #19 : IOError: the file is not EDF(+) or BDF(+) compliant (Label)
- #20 : access subset of data

PyEDFlib 0.1.11 Release Notes

Contents

- *PyEDFlib 0.1.11 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.11*

** Pull requests for v0.1.11*

- Bug fixes

Authors

- Holger Nahrstaedt
- Steffen Heimes +

A total of 2 people contributed to this release. People with a “+” by their names contributed a patch for the first time. This list of names is automatically generated, and may not be fully complete.

Issues closed for v0.1.11

- #21 : Cannot open file containing accented character on Windows

Pull requests for v0.1.11

- #23: Fixed Typo edfreader and improve edfwriter

PyEDFlib 0.1.12 Release Notes

Contents

- *PyEDFlib 0.1.12 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.12*
 - * *Pull requests for v0.1.12*

- small bug fixes

Authors

- Holger Nahrstaedt

Issues closed for v0.1.12

- #24 : Issues reading EDF's
- #25 : file_info_long() method is broken.

Pull requests for v0.1.12

PyEDFlib 0.1.15 Release Notes

Contents

- *PyEDFlib 0.1.15 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.15*
 - * *Pull requests for v0.1.15*

- Fix Documentation (Add class functions)
- Add build for python 3.7
- add close() on EdfReader

Authors

- Holger Nahrstaedt
- skjerns

Issues closed for v0.1.15

- Fix issue #48 - DOC
- Fix issue #60 - No Windows binaries for python 3.7 on PyPI
- Fix issue #62 - Install via pip fails on Windows (no file stdio.h)
- Fix issue #42 - Problem with setGender
- Fix issue #43 - Writer as context manager

Pull requests for v0.1.15

- PR #49 - Added option to get digital samples

PyEDFlib 0.1.16 Release Notes

Contents

- *PyEDFlib 0.1.16 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.16*
 - * *Pull requests for v0.1.16*

- Add high-level functions

Authors

- Holger Nahrstaedt
- skjerns

Issues closed for v0.1.16

- Fix issue #64 - request to provide a sample script to de-identify edf

Pull requests for v0.1.16

- PR #69 - Fix docstring of method setDigitalMaximum
- PR #66 - Add support for PEP 517 & 518
- PR #65 - Added high-level functions

PyEDFlib 0.1.17 Release Notes

Contents

- *PyEDFlib 0.1.17 Release Notes*
 - *Authors*

- Fix readme
- Fix docs
- Add python 3.8 to CI

Authors

- Holger Nahrstaedt

PyEDFlib 0.1.19 Release Notes

Contents

- *PyEDFlib 0.1.19 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.19*
 - * *Pull requests for v0.1.19*

Authors

- Holger Nahrstaedt
- skjerns
- LucaCerina

Issues closed for v0.1.19

- Fix issue #87 - Gender and patient name can't be properly anonymized

Pull requests for v0.1.19

- PR #85 - Add subsecond precision
- PR #89 - Assert that physical_min is different from physical_max
- PR #90 - Add support for gender removal

PyEDFlib 0.1.19 Release Notes

Contents

- *PyEDFlib 0.1.19 Release Notes*
 - *Authors*
 - * *Issues closed for v0.1.19*
 - * *Pull requests for v0.1.19*

Authors

- Holger Nahrstaedt
- skjerns
- LucaCerina

Issues closed for v0.1.19

- Fix issue #87 - Gender and patient name can't be properly anonymized

Pull requests for v0.1.19

- PR #85 - Add subsecond precision
- PR #89 - Assert that physical_min is different from physical_max
- PR #90 - Add support for gender removal

5.5 Indices and tables

- [genindex](#)
- [search](#)

A

`anonymize_edf()` (in module *pyedflib.highlevel*), 28

B

`blockWriteDigitalSamples()` (*pyedflib.EdfWriter* method), 12

`blockWriteDigitalShortSamples()` (*pyedflib.EdfWriter* method), 12

`blockWritePhysicalSamples()` (*pyedflib.EdfWriter* method), 12

C

`change_polarity()` (in module *pyedflib.highlevel*), 27

`close()` (*pyedflib.EdfReader* method), 20

`close()` (*pyedflib.EdfWriter* method), 12

`compare_edf()` (in module *pyedflib.highlevel*), 29

D

`dig2phys()` (in module *pyedflib.highlevel*), 32

`drop_channels()` (in module *pyedflib.highlevel*), 28

E

`EdfReader` (class in *pyedflib*), 18

`EdfWriter` (class in *pyedflib*), 11

F

`file_info()` (*pyedflib.EdfReader* method), 20

`file_info_long()` (*pyedflib.EdfReader* method), 20

G

`getAdmincode()` (*pyedflib.EdfReader* method), 20

`getBirthdate()` (*pyedflib.EdfReader* method), 20

`getDigitalMaximum()` (*pyedflib.EdfReader* method), 21

`getDigitalMinimum()` (*pyedflib.EdfReader* method), 21

`getEquipment()` (*pyedflib.EdfReader* method), 21

`getFileDuration()` (*pyedflib.EdfReader* method), 22

`getGender()` (*pyedflib.EdfReader* method), 22

`getHeader()` (*pyedflib.EdfReader* method), 22

`getLabel()` (*pyedflib.EdfReader* method), 22

`getNSamples()` (*pyedflib.EdfReader* method), 23

`getPatientAdditional()` (*pyedflib.EdfReader* method), 23

`getPatientCode()` (*pyedflib.EdfReader* method), 23

`getPatientName()` (*pyedflib.EdfReader* method), 23

`getPhysicalDimension()` (*pyedflib.EdfReader* method), 23

`getPhysicalMaximum()` (*pyedflib.EdfReader* method), 24

`getPhysicalMinimum()` (*pyedflib.EdfReader* method), 24

`getPrefilter()` (*pyedflib.EdfReader* method), 24

`getRecordingAdditional()` (*pyedflib.EdfReader* method), 25

`getSampleFrequencies()` (*pyedflib.EdfReader* method), 25

`getSampleFrequency()` (*pyedflib.EdfReader* method), 25

`getSignalHeader()` (*pyedflib.EdfReader* method), 25

`getSignalHeaders()` (*pyedflib.EdfReader* method), 25

`getSignalLabels()` (*pyedflib.EdfReader* method), 26

`getStartdatetime()` (*pyedflib.EdfReader* method), 26

`getTechnician()` (*pyedflib.EdfReader* method), 26

`getTransducer()` (*pyedflib.EdfReader* method), 26

M

`make_header()` (in module *pyedflib.highlevel*), 31

`make_signal_header()` (in module *pyedflib.highlevel*), 31

`make_signal_headers()` (in module *pyedflib.highlevel*), 30

P

`phys2dig()` (in module `pyedflib.highlevel`), [32](#)

R

`read_edf()` (in module `pyedflib.highlevel`), [30](#)

`read_edf_header()` (in module `pyedflib.highlevel`), [29](#)

`readAnnotations()` (`pyedflib.EdfReader` method), [27](#)

`readSignal()` (`pyedflib.EdfReader` method), [27](#)

`rename_channels()` (in module `pyedflib.highlevel`), [28](#)

S

`set_number_of_annotation_signals()`
(`pyedflib.EdfWriter` method), [17](#)

`setAdmincode()` (`pyedflib.EdfWriter` method), [13](#)

`setBirthdate()` (`pyedflib.EdfWriter` method), [13](#)

`setDatarecordDuration()` (`pyedflib.EdfWriter`
method), [13](#)

`setDigitalMaximum()` (`pyedflib.EdfWriter`
method), [13](#)

`setDigitalMinimum()` (`pyedflib.EdfWriter`
method), [14](#)

`setEquipment()` (`pyedflib.EdfWriter` method), [14](#)

`setGender()` (`pyedflib.EdfWriter` method), [14](#)

`setHeader()` (`pyedflib.EdfWriter` method), [14](#)

`setLabel()` (`pyedflib.EdfWriter` method), [14](#)

`setPatientAdditional()` (`pyedflib.EdfWriter`
method), [14](#)

`setPatientCode()` (`pyedflib.EdfWriter` method), [15](#)

`setPatientName()` (`pyedflib.EdfWriter` method), [15](#)

`setPhysicalDimension()` (`pyedflib.EdfWriter`
method), [15](#)

`setPhysicalMaximum()` (`pyedflib.EdfWriter`
method), [15](#)

`setPhysicalMinimum()` (`pyedflib.EdfWriter`
method), [15](#)

`setPrefilter()` (`pyedflib.EdfWriter` method), [16](#)

`setRecordingAdditional()` (`pyedflib.EdfWriter`
method), [16](#)

`setSamplefrequency()` (`pyedflib.EdfWriter`
method), [16](#)

`setSignalHeader()` (`pyedflib.EdfWriter` method), [16](#)

`setSignalHeaders()` (`pyedflib.EdfWriter` method), [16](#)

`setStartdatetime()` (`pyedflib.EdfWriter` method), [17](#)

`setTechnician()` (`pyedflib.EdfWriter` method), [17](#)

`setTransducer()` (`pyedflib.EdfWriter` method), [17](#)

U

`update_header()` (`pyedflib.EdfWriter` method), [17](#)

W

`write_edf()` (in module `pyedflib.highlevel`), [29](#)

`write_edf_quick()` (in module `pyedflib.highlevel`), [29](#)

`writeAnnotation()` (`pyedflib.EdfWriter` method), [17](#)

`writeDigitalSamples()` (`pyedflib.EdfWriter`
method), [17](#)

`writeDigitalShortSamples()` (`pyed-
flib.EdfWriter` method), [17](#)

`writePhysicalSamples()` (`pyedflib.EdfWriter`
method), [17](#)

`writeSamples()` (`pyedflib.EdfWriter` method), [18](#)