wfdb Documentation

Release 2.2.0

Chen Xie, Julien Dubiel

Apr 29, 2020

Contents

CHAPTER 1

Introduction

The native Python waveform-database (WFDB) package. A library of tools for reading, writing, and processing WFDB signals and annotations.

Core components of this package are based on the original WFDB specifications. This package does not contain the exact same functionality as the original WFDB package. It aims to implement as many of its core features as possible, with user-friendly APIs. Additional useful physiological signal-processing tools are added over time.

CHAPTER 2

Development

The development repository is hosted at: https://github.com/MIT-LCP/wfdb-python

The package is to be expanded with physiological signal-processing tools, and general improvements. Development is made for Python 2.7 and 3.5+ only.

CHAPTER 3

API Reference

The exact API of all accessible functions and classes, as given by the docstrings, grouped by subpackage:

3.1 io

The input/output subpackage contains classes used to represent WFDB objects, and functions to read, write, and download WFDB files.

3.1.1 WFDB Records

wfdb.io.**rdrecord**(*record_name*, *sampfrom=0*, *sampto=None*, *channels=None*, *physical=True*, *pb_dir=None*, *m2s=True*, *smooth_frames=True*, *ignore_skew=False*, *return_res=64*, *force_channels=True*, *channel_names=None*, *warn_empty=False*)

Read a WFDB record and return the signal and record descriptors as attributes in a Record or MultiRecord object.

record_name [str] The name of the WFDB record to be read, without any file extensions. If the argument contains any path delimiter characters, the argument will be interpreted as PATH/BASE_RECORD. Both relative and absolute paths are accepted. If the *pb_dir* parameter is set, this parameter should contain just the base record name, and the files fill be searched for remotely. Otherwise, the data files will be searched for in the local path.

sampfrom [int, optional] The starting sample number to read for all channels.

sampto [int, or 'end', optional] The sample number at which to stop reading for all channels. Reads the entire duration by default.

channels [list, optional] List of integer indices specifying the channels to be read. Reads all channels by default.

physical [bool, optional] Specifies whether to return signals in physical units in the *p_signal* field (True), or digital units in the *d_signal* field (False).

pb_dir [str, optional] Option used to stream data from Physiobank. The Physiobank database directory from which to find the required record files. eg. For record '100' in 'http://physionet.org/physiobank/database/mitdb' pb_dir='mitdb'.

m2s [bool, optional] Used when reading multi-segment records. Specifies whether to directly return a wfdb MultiRecord object (False), or to convert it into and return a wfdb Record object (True).

smooth_frames [bool, optional] Used when reading records with signals having multiple samples per frame. Specifies whether to smooth the samples in signals with more than one sample per frame and return an (MxN) uniform numpy array as the *d_signal* or *p_signal* field (True), or to return a list of 1d numpy arrays containing every expanded sample as the *e_d_signal* or *e_p_signal* field (False).

ignore_skew [bool, optional] Used when reading records with at least one skewed signal. Specifies whether to apply the skew to align the signals in the output variable (False), or to ignore the skew field and load in all values contained in the dat files unaligned (True).

return_res [int, optional] The numpy array dtype of the returned signals. Options are: 64, 32, 16, and 8, where the value represents the numpy int or float dtype. Note that the value cannot be 8 when physical is True since there is no float8 format.

force_channels [bool, optional] Used when reading multi-segment variable layout records. Whether to update the layout specification record, and the converted Record object if *m2s* is True, to match the input *channels* argument, or to omit channels in which no read segment contains the signals.

channel_names [list, optional] List of channel names to return. If this parameter is specified, it takes precedence over *channels*.

warn_empty [bool, optional] Whether to display a warning if the specified channel indices or names are not contained in the record, and no signal is returned.

record [Record or MultiRecord] The wfdb Record or MultiRecord object representing the contents of the record read.

If a signal range or channel selection is specified when calling this function, the resulting attributes of the returned object will be set to reflect the section of the record that is actually read, rather than necessarily the entire record. For example, if *channels=[0, 1, 2]* is specified when reading a 12 channel record, the 'n_sig' attribute will be 3, not 12.

The *rdsamp* function exists as a simple alternative to *rdrecord* for the common purpose of extracting the physical signals and a few important descriptor fields.

```
>>> record = wfdb.rdrecord('sample-data/test01_00s', sampfrom=800,
                           channels=[1, 3])
```

wfdb.io.**rdsamp**(*record_name*, *sampfrom=0*, *sampto=None*, *channels=None*, *pb_dir=None*, *channel_names=None*, *warn_empty=False*)

Read a WFDB record, and return the physical signals and a few important descriptor fields.

record_name [str] The name of the WFDB record to be read (without any file extensions). If the argument contains any path delimiter characters, the argument will be interpreted as PATH/baserecord and the data files will be searched for in the local path.

sampfrom [int, optional] The starting sample number to read for all channels.

sampto [int, or 'end', optional] The sample number at which to stop reading for all channels. Reads the entire duration by default.

channels [list, optional] List of integer indices specifying the channels to be read. Reads all channels by default.

pb_dir [str, optional] Option used to stream data from Physiobank. The Physiobank database directory from which to find the required record files. eg. For record '100' in 'http://physionet.org/physiobank/database/mitdb' pb_dir='mitdb'.

channel_names [list, optional] List of channel names to return. If this parameter is specified, it takes precedence over *channels*.

warn_empty [bool, optional] Whether to display a warning if the specified channel indices or names are not contained in the record, and no signal is returned.

signals [numpy array] A 2d numpy array storing the physical signals from the record.

fields [dict] A dictionary containing several key attributes of the read record:

- fs: The sampling frequency of the record
- units: The units for each channel
- sig_name: The signal name for each channel
- comments: Any comments written in the header

If a signal range or channel selection is specified when calling this function, the resulting attributes of the returned object will be set to reflect the section of the record that is actually read, rather than necessarily the entire record. For example, if *channels=[0, 1, 2]* is specified when reading a 12 channel record, the 'n_sig' attribute will be 3, not 12.

The *rdrecord* function is the base function upon which this one is built. It returns all attributes present, along with the signals, as attributes in a *Record* object. The function, along with the returned data type, has more options than *rdsamp* for users who wish to more directly manipulate WFDB content.

```
>>> signals, fields = wfdb.rdsamp('sample-data/test01_00s',
                                  sampfrom=800,
                                  channel =[1,3])
```

wfdb.io.**wrsamp**(*record_name*, *fs*, *units*, *sig_name*, *p_signal=None*, *d_signal=None*, *fmt=None*, *adc_gain=None*, *baseline=None*, *comments=None*, *base_time=None*, *base_date=None*, *write_dir=''*)

Write a single segment WFDB record, creating a WFDB header file and any associated dat files.

record_name [str] The string name of the WFDB record to be written (without any file extensions).

fs [int, or float] The sampling frequency of the record.

units [list] A list of strings giving the units of each signal channel.

sig_name : A list of strings giving the signal name of each signal channel.

p_signal [numpy array, optional] An (MxN) 2d numpy array, where M is the signal length. Gives the physical signal values intended to be written. Either p_signal or d_signal must be set, but not both. If p_signal is set, this method will use it to perform analogue-digital conversion, writing the resultant digital values to the dat file(s). If fmt is set, gain and baseline must be set or unset together. If fmt is unset, gain and baseline must both be unset.

d_signal [numpy array, optional] An (MxN) 2d numpy array, where M is the signal length. Gives the digital signal values intended to be directly written to the dat file(s). The dtype must be an integer type. Either p_signal or d_signal must be set, but not both. In addition, if d_signal is set, fmt, gain and baseline must also all be set.

fmt [list, optional] A list of strings giving the WFDB format of each file used to store each channel. Accepted formats are: '80','212","16','24', and '32'. There are other WFDB formats as specified by: https://www.

physionet.org/physiotools/wag/signal-5.htm but this library will not write (though it will read) those file types.

adc_gain [list, optional] A list of numbers specifying the ADC gain.

baseline [list, optional] A list of integers specifying the digital baseline.

comments [list, optional] A list of string comments to be written to the header file.

base_time [str, optional] A string of the record's start time in 24h 'HH:MM:SS(.ms)' format.

base_date [str, optional] A string of the record's start date in 'DD/MM/YYYY' format.

write_dir [str, optional] The directory in which to write the files.

This is a gateway function, written as a simple method to write WFDB record files using the most common parameters. Therefore not all WFDB fields can be set via this function.

For more control over attributes, create a *Record* object, manually set its attributes, and call its *wrsamp* instance method. If you choose this more advanced method, see also the *set_defaults*, *set_d_features*, and *set_p_features* instance methods to help populate attributes.

```
>>> # Read part of a record from Physiobank
>>> signals, fields = wfdb.rdsamp('a103l', sampfrom=50000, channels=[0,1],
                                  pb_dir='challenge/2015/training')
>>> # Write a local WFDB record (manually inserting fields)
>>> wfdb.wrsamp('ecgrecord', fs = 250, units=['mV', 'mV'],
                sig_name=['I', 'II'], p_signal=signals, fmt=['16', '16'])
```

class wfdb.io.**Record**(*p_signal=None*, *d_signal=None*, *e_p_signal=None*, *e_d_signal=None*, *record_name=None*, *n_sig=None*, *fs=None*, *counter_freq=None*, *base_counter=None*, *sig_len=None*, *base_time=None*, *base_date=None*, *file_name=None*, *fmt=None*, *samps_per_frame=None*, *skew=None*, *byte_offset=None*, *adc_gain=None*, *baseline=None*, *units=None*, *adc_res=None*, *adc_zero=None*, *init_value=None*, *checksum=None*, *block_size=None*, *sig_name=None*, *comments=None*)

The class representing single segment WFDB records.

Record objects can be created using the initializer, by reading a WFDB header with *rdheader*, or a WFDB record (header and associated dat files) with *rdrecord*.

The attributes of the Record object give information about the record as specified by: https://www.physionet.org/physiotools/wag/header-5.htm

In addition, the d_signal and p_signal attributes store the digital and physical signals of WFDB records with at least one channel.

```
>>> record = wfdb.Record(record_name='r1', fs=250, n_sig=2, sig_len=1000,
                         file_name=['r1.dat','r1.dat'])
```

adc(*expanded=False*, *inplace=False*)

Performs analogue to digital conversion of the physical signal stored in p_signal if expanded is False, or e_p_signal if expanded is True.

The p_signal/e_p_signal, fmt, gain, and baseline fields must all be valid.

If inplace is True, the adc will be performed inplace on the variable, the d_signal/e_d_signal attribute will be set, and the p_signal/e_p_signal field will be set to None.

expanded [bool, optional] Whether to transform the *e_p_signal* attribute (True) or the *p_signal* attribute (False).

inplace [bool, optional] Whether to automatically set the object's corresponding digital signal attribute and set the physical signal attribute to None (True), or to return the converted signal as a separate variable without changing the original physical signal attribute (False).

d_signal [numpy array, optional] The digital conversion of the signal. Either a 2d numpy array or a list of 1d numpy arrays.

```
>>> import wfdb
>>> record = wfdb.rdsamp('sample-data/100')
>>> d_signal = record.adc()
>>> record.adc(inplace=True)
>>> record.dac(inplace=True)
```

dac(*expanded=False*, *return_res=64*, *inplace=False*)

Performs the digital to analogue conversion of the signal stored in *d_signal* if expanded is False, or *e_d_signal* if expanded is True.

The d_signal/e_d_signal, fmt, gain, and baseline fields must all be valid.

If inplace is True, the dac will be performed inplace on the variable, the p_signal/e_p_signal attribute will be set, and the d_signal/e_d_signal field will be set to None.

expanded [bool, optional] Whether to transform the *e_d_signal attribute* (True) or the *d_signal* attribute (False).

inplace [bool, optional] Whether to automatically set the object's corresponding physical signal attribute and set the digital signal attribute to None (True), or to return the converted signal as a separate variable without changing the original digital signal attribute (False).

p_signal [numpy array, optional] The physical conversion of the signal. Either a 2d numpy array or a list of 1d numpy arrays.

```
>>> import wfdb
>>> record = wfdb.rdsamp('sample-data/100', physical=False)
>>> p_signal = record.dac()
>>> record.dac(inplace=True)
>>> record.adc(inplace=True)
```

wrsamp(*expanded=False*, *write_dir=''*)

Write a wfdb header file and any associated dat files from this object.

expanded [bool, optional] Whether to write the expanded signal (e_d_signal) instead of the uniform signal (d_signal).

write_dir [str, optional] The directory in which to write the files.

class wfdb.io.**MultiRecord**(*segments=None*, *layout=None*, *record_name=None*, *n_sig=None*, *fs=None*, *counter_freq=None*, *base_counter=None*, *sig_len=None*, *base_time=None*, *base_date=None*, *seg_name=None*, *seg_len=None*, *comments=None*, *sig_name=None*, *sig_segments=None*)

The class representing multi-segment WFDB records.

MultiRecord objects can be created using the initializer, or by reading a multi-segment WFDB record using 'rdrecord' with the *m2s* (multi to single) input parameter set to False.

The attributes of the MultiRecord object give information about the entire record as specified by: https://www.physionet.org/physiotools/wag/header-5.htm

In addition, the *segments* parameter is a list of Record objects representing each individual segment, or None representing empty segments, of the entire multi-segment record.

Notably, this class has no attribute representing the signals as a whole. The 'multi_to_single' instance method can be called on MultiRecord objects to return a single segment representation of the record as a Record object. The resulting Record object will have its 'p_signal' field set.

```
>>> record_m = wfdb.MultiRecord(record_name='rm', fs=50, n_sig=8,
                                sig_len=9999, seg_name=['rm_1', '~', rm_2'],
                                seg_len=[800, 200, 900])
>>> # Get a MultiRecord object
>>> record_s = wfdb.rdsamp('s00001-2896-10-10-00-31', m2s=False)
>>> # Turn it into a
>>> record_s = record_s.multi_to_single()
```

record_s initially stores a *MultiRecord* object, and is then converted into a *Record* object.

multi_to_single(*physical*, *return_res=64*)

Create a Record object from the MultiRecord object. All signal segments will be combined into the new object's *p_signal* or *d_signal* field. For digital format, the signals must have the same storage format, baseline, and adc_gain in all segments.

physical [bool] Whether to convert the physical or digital signal.

return_res [int, optional] The return resolution of the *p_signal* field. Options are: 64, 32, and 16.

record [wfdb Record] The single segment record created.

3.1.2 WFDB Anotations

wfdb.io.**rdann**(*record_name, extension, sampfrom=0, sampto=None, shift_samps=False, pb_dir=None, return_label_elements=['symbol'], summarize_labels=False*)

Read a WFDB annotation file record_name.extension and return an Annotation object.

record_name [str] The record name of the WFDB annotation file. ie. for file '100.atr', record_name='100'.

extension [str] The annotatator extension of the annotation file. ie. for file '100.atr', extension='atr'.

sampfrom [int, optional] The minimum sample number for annotations to be returned.

sampto [int, optional] The maximum sample number for annotations to be returned.

shift_samps [bool, optional] Specifies whether to return the sample indices relative to *sampfrom* (True), or sample 0 (False).

pb_dir [str, optional] Option used to stream data from Physiobank. The Physiobank database directory from which to find the required annotation file. eg. For record '100' in 'http://physionet.org/physiobank/database/mitdb': pb_dir='mitdb'.

return_label_elements [list, optional] The label elements that are to be returned from reading the annotation file. A list with at least one of the following options: 'symbol', 'label_store', 'description'.

summarize_labels [bool, optional] If True, assign a summary table of the set of annotation labels contained in the file to the 'contained_labels' attribute of the returned object. This table will contain the columns: ['label_store', 'symbol', 'description', 'n_occurences']

annotation [Annotation] The Annotation object. Call help(wfdb.Annotation) for the attribute descriptions.

For every annotation sample, the annotation file explictly stores the 'sample' and 'symbol' fields, but not necessarily the others. When reading annotation files using this function, fields which are not stored in the file will either take their default values of 0 or None, or will be carried over from their previous values if any.

```
>>> ann = wfdb.rdann('sample-data/100', 'atr', sampto=300000)
```

wfdb.io.**wrann**(*record_name*, *extension*, *sample*, *symbol=None*, *subtype=None*, *chan=None*, *num=None*, *aux_note=None*, *label_store=None*, *fs=None*, *custom_labels=None*, *write_dir=''*)

Write a WFDB annotation file.

Specify at least the following:

- The record name of the WFDB record (record_name)
- The annotation file extension (extension)
- The annotation locations in samples relative to the beginning of the record (sample)
- Either the numerical values used to store the labels (*label_store*), or more commonly, the display symbols of each label (*symbol*).

record_name [str] The string name of the WFDB record to be written (without any file extensions).

extension [str] The string annotation file extension.

sample [numpy array] A numpy array containing the annotation locations in samples relative to the beginning of the record.

symbol [list, or numpy array, optional] The symbols used to display the annotation labels. List or numpy array. If this field is present, *label_store* must not be present.

subtype [numpy array, optional] A numpy array containing the marked class/category of each annotation.

chan [numpy array, optional] A numpy array containing the signal channel associated with each annotation.

num [numpy array, optional] A numpy array containing the labelled annotation number for each annotation.

aux_note [list, optional] A list containing the auxiliary information string (or None for annotations without notes) for each annotation.

label_store [numpy array, optional] A numpy array containing the integer values used to store the annotation labels. If this field is present, *symbol* must not be present.

fs [int, or float, optional] The numerical sampling frequency of the record to be written to the file.

custom_labels [pandas dataframe, optional] The map of custom defined annotation labels used for this annotation, in addition to the standard WFDB annotation labels. Custom labels are defined by two or three fields:

- The integer values used to store custom annotation labels in the file (optional)
- Their short display symbols
- Their long descriptions.

This input argument may come in four formats:

1. A pandas.DataFrame object with columns: ['label_store', 'symbol', 'description']
2. A pandas.DataFrame object with columns: ['symbol', 'description'] If this option is chosen, label_store values are automatically chosen.
3. A list or tuple of tuple triplets, with triplet elements representing: (label_store, symbol, description).
4. A list or tuple of tuple pairs, with pair elements representing: (symbol, description). If this option is chosen, label_store values are automatically chosen.

If the *label_store* field is given for this function, and *custom_labels* is defined, *custom_labels* must contain *label_store* in its mapping. ie. it must come in format 1 or 3 above.

write_dir [str, optional] The directory in which to write the annotation file

This is a gateway function, written as a simple way to write WFDB annotation files without needing to explicity create an Annotation object. You may also create an Annotation object, manually set its attributes, and call its *wrann* instance method.

Each annotation stored in a WFDB annotation file contains a sample field and a label field. All other fields may or may not be present.

```
>>> # Read an annotation as an Annotation object
>>> annotation = wfdb.rdann('b001', 'atr', pb_dir='cebsdb')
>>> # Write a copy of the annotation file
>>> wfdb.wrann('b001', 'cpy', annotation.sample, annotation.symbol)
```

wfdb.io.**show_ann_labels**()
Display the standard wfdb annotation label mapping.

```
>>> show_ann_labels()
```

wfdb.io.**show_ann_classes**()
Display the standard wfdb annotation classes

```
>>> show_ann_classes()
```

class wfdb.io.**Annotation**(*record_name*, *extension*, *sample*, *symbol=None*, *subtype=None*, *chan=None*, *num=None*, *aux_note=None*, *fs=None*, *label_store=None*, *description=None*, *custom_labels=None*, *contained_labels=None*)
The class representing WFDB annotations.

Annotation objects can be created using the initializer, or by reading a WFDB annotation file with *rdann*.

The attributes of the Annotation object give information about the annotation as specified by: https://www.physionet.org/physiotools/wag/annot-5.htm

Call *show_ann_labels()* to see the list of standard annotation codes. Any text used to label annotations that are not one of these codes should go in the 'aux_note' field rather than the 'sym' field.

```
>>> ann1 = wfdb.Annotation(record_name='rec1', extension='atr',
                           sample=[10,20,400], symbol=['N','N','['],
                           aux_note=[None, None, 'Serious Vfib'])
```

wrann(*write_fs=False*, *write_dir=''*)
Write a WFDB annotation file from this object.

write_fs [bool, optional] Whether to write the *fs* attribute to the file.

3.1.3 Downloading

wfdb.io.**get_dbs**()
Get a list of all the Physiobank databases available.

```
>>> dbs = get_dbs()
```

wfdb.io.**get_record_list**(*db_dir*, *records='all'*)
Get a list of records belonging to a database.

db_dir [str] The database directory, usually the same as the database slug. The location to look for a RECORDS file.

records [list, optional] A Option used when this function acts as a helper function. Leave as default 'all' to get all records.

```
>>> wfdb.get_record_list('mitdb')
```

wfdb.io.**dl_database**(*db_dir*, *dl_dir*, *records='all'*, *annotators='all'*, *keep_subdirs=True*, *overwrite=False*)

Download WFDB record (and optionally annotation) files from a Physiobank database. The database must contain a 'RECORDS' file in its base directory which lists its WFDB records.

db_dir [str] The Physiobank database directory to download. eg. For database: 'http://physionet.org/physiobank/database/mitdb', db_dir='mitdb'.

dl_dir [str] The full local directory path in which to download the files.

records [list, or 'all', optional] A list of strings specifying the WFDB records to download. Leave as 'all' to download all records listed in the database's RECORDS file. eg. records=['test01_00s', test02_45s] for database: https://physionet.org/physiobank/database/macecgdb/

annotators [list, 'all', or None, optional] A list of strings specifying the WFDB annotation file types to download along with the record files. Is either None to skip downloading any annotations, 'all' to download all annotation types as specified by the ANNOTATORS file, or a list of strings which each specify an annotation extension. eg. annotators = ['anI'] for database: https://physionet.org/physiobank/database/prcp/

keep_subdirs [bool, optional] Whether to keep the relative subdirectories of downloaded files as they are organized in Physiobank (True), or to download all files into the same base directory (False).

overwrite [bool, optional] If True, all files will be redownloaded regardless. If False, existing files with the same name and relative subdirectory will be checked. If the local file is the same size as the online file, the download is skipped. If the local file is larger, it will be deleted and the file will be redownloaded. If the local file is smaller, the file will be assumed to be partially downloaded and the remaining bytes will be downloaded and appended.

```
>>> wfdb.dl_database('ahadb', os.getcwd())
```

wfdb.io.**dl_files**(*db*, *dl_dir*, *files*, *keep_subdirs=True*, *overwrite=False*)

Download specified files from a Physiobank database.

db [str] The Physiobank database directory to download. eg. For database: 'http://physionet.org/physiobank/database/mitdb', db='mitdb'.

dl_dir [str] The full local directory path in which to download the files.

files [list] A list of strings specifying the file names to download relative to the database base directory.

keep_subdirs [bool, optional] Whether to keep the relative subdirectories of downloaded files as they are organized in Physiobank (True), or to download all files into the same base directory (False).

overwrite [bool, optional] If True, all files will be redownloaded regardless. If False, existing files with the same name and relative subdirectory will be checked. If the local file is the same size as the online file, the download is skipped. If the local file is larger, it will be deleted and the file will be redownloaded. If the local file is smaller, the file will be assumed to be partially downloaded and the remaining bytes will be downloaded and appended.

```
>>> wfdb.dl_files('ahadb', os.getcwd(),
                  ['STAFF-Studies-bibliography-2016.pdf', 'data/001a.hea',
                  'data/001a.dat'])
```

wfdb.io.**set_db_index_url**(*db_index_url='http://physionet.org/physiobank/database/'*)
Set the database index url to a custom value, to stream remote files from another location.

db_index_url [str, optional] The desired new database index url. Leave as default to reset to the physiobank index url.

3.2 plot

The plot subpackage contains tools for plotting signals and annotations.

wfdb.plot.**plot_items**(*signal=None, ann_samp=None, ann_sym=None, fs=None, time_units='samples', sig_name=None, sig_units=None, ylabel=None, title=None, sig_style=[''], ann_style=['r*'], ecg_grids=[], figsize=None, return_fig=False*)
Subplot individual channels of signals and/or annotations.

signal [1d or 2d numpy array, optional] The uniformly sampled signal to be plotted. If signal.ndim is 1, it is assumed to be a one channel signal. If it is 2, axes 0 and 1, must represent time and channel number respectively.

ann_samp: list, optional A list of annotation locations to plot, with each list item corresponding to a different channel. List items may be:

- 1d numpy array, with values representing sample indices. Empty arrays are skipped.
- list, with values representing sample indices. Empty lists are skipped.
- None. For channels in which nothing is to be plotted.

If *signal* is defined, the annotation locations will be overlaid on the signals, with the list index corresponding to the signal channel. The length of *annotation* does not have to match the number of channels of *signal*.

ann_sym: list, optional A list of annotation symbols to plot, with each list item corresponding to a different channel. List items should be lists of strings. The symbols are plotted over the corresponding *ann_samp* index locations.

fs [int or float, optional] The sampling frequency of the signals and/or annotations. Used to calculate time intervals if *time_units* is not 'samples'. Also required for plotting ecg grids.

time_units [str, optional] The x axis unit. Allowed options are: 'samples', 'seconds', 'minutes', and 'hours'.

sig_name [list, optional] A list of strings specifying the signal names. Used with *sig_units* to form y labels, if *ylabel* is not set.

sig_units [list, optional] A list of strings specifying the units of each signal channel. Used with *sig_name* to form y labels, if *ylabel* is not set. This parameter is required for plotting ecg grids.

ylabel [list, optional] A list of strings specifying the final y labels. If this option is present, *sig_name* and *sig_units* will not be used for labels.

title [str, optional] The title of the graph.

sig_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each signal channel. The list length should match the number of signal channels. If the list has a length of 1, the style will be used for all channels.

ann_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each annotation channel. If the list has a length of 1, the style will be used for all channels.

ecg_grids [list, optional] A list of integers specifying channels in which to plot ecg grids. May also be set to 'all' for all channels. Major grids at 0.5mV, and minor grids at 0.125mV. All channels to be plotted with grids must have *sig_units* equal to 'uV', 'mV', or 'V'.

figsize [tuple, optional] Tuple pair specifying the width, and height of the figure. It is the 'figsize' argument passed into matplotlib.pyplot's *figure* function.

return_fig [bool, optional] Whether the figure is to be returned as an output argument.

figure [matplotlib figure, optional] The matplotlib figure generated. Only returned if the 'return_fig' parameter is set to True.

```
>>> record = wfdb.rdrecord('sample-data/100', sampto=3000)
>>> ann = wfdb.rdann('sample-data/100', 'atr', sampto=3000)
```

```
>>> wfdb.plot_items(signal=record.p_signal,
                    annotation=[ann.sample, ann.sample],
                    title='MIT-BIH Record 100', time_units='seconds',
                    figsize=(10,4), ecg_grids='all')
```

wfdb.plot.**plot_wfdb**(*record=None, annotation=None, plot_sym=False, time_units='samples', title=None, sig_style=[''], ann_style=['r*'], ecg_grids=[], figsize=None, return_fig=False*)

Subplot individual channels of a wfdb record and/or annotation.

This function implements the base functionality of the *plot_items* function, while allowing direct input of wfdb objects.

If the record object is input, the function will extract from it:

- signal values, from the *p_signal* (priority) or *d_signal* attribute
- sampling frequency, from the *fs* attribute
- signal names, from the *sig_name* attribute
- signal units, from the *units* attribute

If the annotation object is input, the function will extract from it:

- sample locations, from the *sample* attribute
- symbols, from the *symbol* attribute
- the annotation channels, from the *chan* attribute
- the sampling frequency, from the *fs* attribute if present, and if fs was not already extracted from the *record* argument.

record [wfdb Record, optional] The Record object to be plotted

annotation [wfdb Annotation, optional] The Annotation object to be plotted

plot_sym [bool, optional] Whether to plot the annotation symbols on the graph.

time_units [str, optional] The x axis unit. Allowed options are: 'samples', 'seconds', 'minutes', and 'hours'.

title [str, optional] The title of the graph.

sig_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each signal channel. The list length should match the number of signal channels. If the list has a length of 1, the style will be used for all channels.

ann_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each annotation channel. The list length should match the number of annotation channels. If the list has a length of 1, the style will be used for all channels.

ecg_grids [list, optional] A list of integers specifying channels in which to plot ecg grids. May also be set to 'all' for all channels. Major grids at 0.5mV, and minor grids at 0.125mV. All channels to be plotted with grids must have *sig_units* equal to 'uV', 'mV', or 'V'.

figsize [tuple, optional] Tuple pair specifying the width, and height of the figure. It is the 'figsize' argument passed into matplotlib.pyplot's *figure* function.

return_fig [bool, optional] Whether the figure is to be returned as an output argument.

figure [matplotlib figure, optional] The matplotlib figure generated. Only returned if the 'return_fig' option is set to True.

```
>>> record = wfdb.rdrecord('sample-data/100', sampto=3000)
>>> annotation = wfdb.rdann('sample-data/100', 'atr', sampto=3000)
```

```
>>> wfdb.plot_wfdb(record=record, annotation=annotation, plot_sym=True
                   time_units='seconds', title='MIT-BIH Record 100',
                   figsize=(10,4), ecg_grids='all')
```

wfdb.plot.**plot_all_records**(*directory=''*)

Plot all wfdb records in a directory (by finding header files), one at a time, until the 'enter' key is pressed.

directory [str, optional] The directory in which to search for WFDB records. Defaults to current working directory.

3.3 processing

The processing subpackage contains signal-processing tools.

3.3.1 Basic Utility

Basic signal processing functions

wfdb.processing.**resample_ann**(*resampled_t*, *ann_sample*)

Compute the new annotation indices

resampled_t [numpy array] Array of signal locations as returned by scipy.signal.resample

ann_sample [numpy array] Array of annotation locations

resampled_ann_sample [numpy array] Array of resampled annotation locations

wfdb.processing.**resample_sig**(*x*, *fs*, *fs_target*)

Resample a signal to a different frequency.

x [numpy array] Array containing the signal

fs [int, or float] The original sampling frequency

fs_target [int, or float] The target frequency

resampled_x [numpy array] Array of the resampled signal values

resampled_t [numpy array] Array of the resampled signal locations

wfdb.processing.**resample_singlechan**(*x*, *ann*, *fs*, *fs_target*)
Resample a single-channel signal with its annotations

x: numpy array The signal array

ann [wfdb Annotation] The wfdb annotation object

fs [int, or float] The original frequency

fs_target [int, or float] The target frequency

resampled_x [numpy array] Array of the resampled signal values

resampled_ann [wfdb Annotation] Annotation containing resampled annotation locations

wfdb.processing.**resample_multichan**(*xs*, *ann*, *fs*, *fs_target*, *resamp_ann_chan=0*)
Resample multiple channels with their annotations

xs: numpy array The signal array

ann [wfdb Annotation] The wfdb annotation object

fs [int, or float] The original frequency

fs_target [int, or float] The target frequency

resample_ann_channel [int, optional] The signal channel used to compute new annotation indices

resampled_xs [numpy array] Array of the resampled signal values

resampled_ann [wfdb Annotation] Annotation containing resampled annotation locations

wfdb.processing.**normalize_bound**(*sig*, *lb=0*, *ub=1*)
Normalize a signal between the lower and upper bound

sig [numpy array] Original signal to be normalized

lb [int, or float] Lower bound

ub [int, or float] Upper bound

x_normalized [numpy array] Normalized signal

wfdb.processing.**get_filter_gain**(*b*, *a*, *f_gain*, *fs*)
Given filter coefficients, return the gain at a particular frequency.

b [list] List of linear filter b coefficients

a [list] List of linear filter a coefficients

f_gain [int or float, optional] The frequency at which to calculate the gain

fs [int or float, optional] The sampling frequency of the system

3.3.2 Heart Rate

wfdb.processing.**compute_hr**(*sig_len*, *qrs_inds*, *fs*)
Compute instantaneous heart rate from peak indices.

sig_len [int] The length of the corresponding signal

qrs_inds [numpy array] The qrs index locations

fs [int, or float] The corresponding signal's sampling frequency.

heart_rate [numpy array] An array of the instantaneous heart rate, with the length of the corresponding signal. Contains numpy.nan where heart rate could not be computed.

3.3.3 Peaks

wfdb.processing.**find_peaks**(*sig*)

Find hard peaks and soft peaks in a signal, defined as follows:

- Hard peak: a peak that is either /or /
- Soft peak: a peak that is either /-*or* -/ In this case we define the middle as the peak

sig [np array] The 1d signal array

hard_peaks [numpy array] Array containing the indices of the hard peaks:

soft_peaks [numpy array] Array containing the indices of the soft peaks

wfdb.processing.**find_local_peaks**(*sig*, *radius*)

Find all local peaks in a signal. A sample is a local peak if it is the largest value within the <radius> samples on its left and right.

In cases where it shares the max value with nearby samples, the middle sample is classified as the local peak.

sig [numpy array] 1d numpy array of the signal.

radius [int] The radius in which to search for defining local maxima.

wfdb.processing.**correct_peaks**(*sig*, *peak_inds*, *search_radius*, *smooth_window_size*, *peak_dir='compare'*)

Adjust a set of detected peaks to coincide with local signal maxima, and

sig [numpy array] The 1d signal array

peak_inds [np array] Array of the original peak indices

max_gap [int] The radius within which the original peaks may be shifted.

smooth_window_size [int] The window size of the moving average filter applied on the signal. Peak distance is calculated on the difference between the original and smoothed signal.

peak_dir [str, optional] The expected peak direction: 'up' or 'down', 'both', or 'compare'.

- If 'up', the peaks will be shifted to local maxima
- If 'down', the peaks will be shifted to local minima
- If 'both', the peaks will be shifted to local maxima of the rectified signal
- If 'compare', the function will try both 'up' and 'down' options, and choose the direction that gives the largest mean distance from the smoothed signal.

corrected_peak_inds [numpy array] Array of the corrected peak indices

3.3.4 QRS Detectors

class wfdb.processing.**XQRS**(*sig*, *fs*, *conf=None*)

The qrs detector class for the xqrs algorithm.

The *XQRS.Conf* class is the configuration class that stores initial parameters for the detection.

The *XQRS.detect* method runs the detection algorithm.

The process works as follows:

- Load the signal and configuration parameters.
- Bandpass filter the signal between 5 and 20 Hz, to get the filtered signal.
- Apply moving wave integration (mwi) with a ricker (Mexican hat) wavelet onto the filtered signal, and save the square of the integrated signal.
- Conduct learning if specified, to initialize running parameters of noise and qrs amplitudes, the qrs detection threshold, and recent rr intervals. If learning is unspecified or fails, use default parameters.
- Run the main detection. Iterate through the local maxima of the mwi signal. For each local maxima:
  - Check if it is a qrs complex. To be classified as a qrs, it must come after the refractory period, cross the qrs detection threshold, and not be classified as a t-wave if it comes close enough to the previous qrs. If successfully classified, update running detection threshold and heart rate parameters.
  - If not a qrs, classify it as a noise peak and update running parameters.
  - Before continuing to the next local maxima, if no qrs was detected within 1.66 times the recent rr interval, perform backsearch qrs detection. This checks previous peaks using a lower qrs detection threshold.

```
>>> import wfdb
>>> from wfdb import processing
```

```
>>> sig, fields = wfdb.rdsamp('sample-data/100', channels=[0])
>>> xqrs = processing.XQRS(sig=sig[:,0], fs=fields['fs'])
>>> xqrs.detect()
```

```
>>> wfdb.plot_items(signal=sig, ann_samp=[xqrs.qrs_inds])
```

class Conf(*hr_init=75*, *hr_max=200*, *hr_min=25*, *qrs_width=0.1*, *qrs_thr_init=0.13*, *qrs_thr_min=0*, *ref_period=0.2*, *t_inspect_period=0.36*)

Initial signal configuration object for this qrs detector

detect(*sampfrom=0*, *sampto='end'*, *learn=True*, *verbose=True*)

Detect qrs locations between two samples.

sampfrom [int, optional] The starting sample number to run the detection on.

sampto [int, optional] The final sample number to run the detection on. Set as 'end' to run on the entire signal.

learn [bool, optional] Whether to apply learning on the signal before running the main detection. If learning fails or is not conducted, the default configuration parameters will be used to initialize these variables. See the *XQRS._learn_init_params* docstring for details.

verbose [bool, optional] Whether to display the stages and outcomes of the detection process.

wfdb.processing.**xqrs_detect**(*sig*, *fs*, *sampfrom=0*, *sampto='end'*, *conf=None*, *learn=True*, *verbose=True*)

Run the 'xqrs' qrs detection algorithm on a signal. See the docstring of the XQRS class for algorithm details.

sig [numpy array] The input ecg signal to apply the qrs detection on.

fs [int or float] The sampling frequency of the input signal.

sampfrom [int, optional] The starting sample number to run the detection on.

sampto : The final sample number to run the detection on. Set as 'end' to run on the entire signal.

conf [XQRS.Conf object, optional] The configuration object specifying signal configuration parameters. See the docstring of the XQRS.Conf class.

learn [bool, optional] Whether to apply learning on the signal before running the main detection. If learning fails or is not conducted, the default configuration parameters will be used to initialize these variables.

verbose [bool, optional] Whether to display the stages and outcomes of the detection process.

qrs_inds [numpy array] The indices of the detected qrs complexes

```
>>> import wfdb
>>> from wfdb import processing
```

```
>>> sig, fields = wfdb.rdsamp('sample-data/100', channels=[0])
>>> qrs_inds = processing.xqrs_detect(sig=sig[:,0], fs=fields['fs'])
```

wfdb.processing.**gqrs_detect**(*sig=None*, *fs=None*, *d_sig=None*, *adc_gain=None*, *adc_zero=None*, *threshold=1.0*, *hr=75*, *RRdelta=0.2*, *RRmin=0.28*, *RRmax=2.4*, *QS=0.07*, *QT=0.35*, *RTmin=0.25*, *RTmax=0.33*, *QRSa=750*, *QRSamin=130*)

Detect qrs locations in a single channel ecg. Functionally, a direct port of the gqrs algorithm from the original wfdb package. Accepts either a physical signal, or a digital signal with known adc_gain and adc_zero.

See the notes below for a summary of the program. This algorithm is not being developed/supported.

sig [1d numpy array, optional] The input physical signal. The detection algorithm which replicates the original, works using digital samples, and this physical option is provided as a convenient interface. If this is the specified input signal, automatic adc is performed using 24 bit precision, to obtain the *d_sig*, *adc_gain*, and *adc_zero* parameters. There may be minor differences in detection results (ie. an occasional 1 sample difference) between using *sig* and *d_sig*. To replicate the exact output of the original gqrs algorithm, use the *d_sig* argument instead.

fs [int, or float] The sampling frequency of the signal.

d_sig [1d numpy array, optional] The input digital signal. If this is the specified input signal rather than *sig*, the *adc_gain* and *adc_zero* parameters must be specified.

adc_gain [int, or float, optional] The analogue to digital gain of the signal (the number of adus per physical unit).

adc_zero: int, optional The value produced by the ADC given a 0 volt input.

threshold [int, or float, optional] The relative amplitude detection threshold. Used to initialize the peak and qrs detection threshold.

hr [int, or float, optional] Typical heart rate, in beats per minute.

RRdelta [int or float, optional] Typical difference between successive RR intervals in seconds.

RRmin [int or float, optional] Minimum RR interval ("refractory period"), in seconds.

RRmax [int or float, optional] Maximum RR interval, in seconds. Thresholds will be adjusted if no peaks are detected within this interval.

QS [int or float, optional] Typical QRS duration, in seconds.

QT [int or float, optional] Typical QT interval, in seconds.

RTmin [int or float, optional] Minimum interval between R and T peaks, in seconds.

RTmax [int or float, optional] Maximum interval between R and T peaks, in seconds.

QRSa [int or float, optional] Typical QRS peak-to-peak amplitude, in microvolts.

QRSamin [int or float, optional] Minimum QRS peak-to-peak amplitude, in microvolts.

qrs_locs [numpy array] Detected qrs locations

This function should not be used for signals with fs <= 50Hz

The algorithm theoretically works as follows:

- Load in configuration parameters. They are used to set/initialize the:
  - allowed rr interval limits (fixed)
  - initial recent rr interval (running)
  - qrs width, used for detection filter widths (fixed)
  - allowed rt interval limits (fixed)
  - initial recent rt interval (running)
  - initial peak amplitude detection threshold (running)
  - initial qrs amplitude detection threshold (running)
  - *Note*: this algorithm does not normalize signal amplitudes, and hence is highly dependent on configuration amplitude parameters.
- Apply trapezoid low-pass filtering to the signal
- Convolve a QRS matched filter with the filtered signal
- Run the learning phase using a calculated signal length: detect qrs and non-qrs peaks as in the main detection phase, without saving the qrs locations. During this phase, running parameters of recent intervals and peak/qrs thresholds are adjusted.
- **Run the detection::** if a sample is bigger than its immediate neighbors and larger than the peak detection threshold, it is a peak.

  if it is further than RRmin from the previous qrs, and is a *primary peak.

  if it is further than 2 standard deviations from the previous qrs, do a backsearch for a missed low amplitude beat

  return the primary peak between the current sample and the previous qrs if any.

  if it surpasses the qrs threshold, it is a qrs complex save the qrs location. update running rr and qrs amplitude parameters. look for the qrs complex's t-wave and mark it if found.

  else if it is not a peak lower the peak detection threshold if the last peak found was more than RRmax ago, and not already at its minimum.

*A peak is secondary if there is a larger peak within its neighborhood (time +- rrmin), or if it has been identified as a T-wave associated with a previous primary peak. A peak is primary if it is largest in its neighborhood, or if the only larger peaks are secondary.

The above describes how the algorithm should theoretically work, but there are bugs which make the program contradict certain parts of its supposed logic. A list of issues from the original c, code and hence this python implementation can be found here:

https://github.com/bemoody/wfdb/issues/17

gqrs will not be supported/developed in this library.

```
>>> import numpy as np
>>> import wfdb
>>> from wfdb import processing
```

```
>>> # Detect using a physical input signal
>>> record = wfdb.rdrecord('sample-data/100', channels=[0])
>>> qrs_locs = processing.gqrs_detect(record.p_signal[:,0], fs=record.fs)
```

```
>>> # Detect using a digital input signal
>>> record_2 = wfdb.rdrecord('sample-data/100', channels=[0], physical=False)
>>> qrs_locs_2 = processing.gqrs_detect(d_sig=record_2.d_signal[:,0],
                                        fs=record_2.fs,
                                        adc_gain=record_2.adc_gain[0],
                                        adc_zero=record_2.adc_zero[0])
```

3.3.5 Annotation Evaluators

class wfdb.processing.**Comparitor**(*ref_sample*, *test_sample*, *window_width*, *signal=None*)

The class to implement and hold comparisons between two sets of annotations.

See methods *compare*, *print_summary* and *plot*.

```
>>> import wfdb
>>> from wfdb import processing
```

```
>>> sig, fields = wfdb.rdsamp('sample-data/100', channels=[0])
>>> ann_ref = wfdb.rdann('sample-data/100','atr')
>>> xqrs = processing.XQRS(sig=sig[:,0], fs=fields['fs'])
>>> xqrs.detect()
```

```
>>> comparitor = processing.Comparitor(ann_ref.sample[1:],
                                       xqrs.qrs_inds,
                                       int(0.1 * fields['fs']),
                                       sig[:,0])
>>> comparitor.compare()
>>> comparitor.print_summary()
>>> comparitor.plot()
```

compare()

Main comparison function

plot(*sig_style=''*, *title=None*, *figsize=None*, *return_fig=False*)

Plot the comparison of two sets of annotations, possibly overlaid on their original signal.

sig_style [str, optional] The matplotlib style of the signal

title [str, optional] The title of the plot

figsize: tuple, optional Tuple pair specifying the width, and height of the figure. It is the'figsize' argument passed into matplotlib.pyplot's *figure* function.

return_fig [bool, optional] Whether the figure is to be returned as an output argument.

print_summary()
Print summary metrics of the annotation comparisons.

wfdb.processing.**compare_annotations**(*ref_sample*, *test_sample*, *window_width*, *signal=None*)
Compare a set of reference annotation locations against a set of test annotation locations.

See the Comparitor class docstring for more information.

ref_sample [1d numpy array] Array of reference sample locations

test_sample [1d numpy array] Array of test sample locations to compare

window_width [int] The maximum absolute difference in sample numbers that is permitted for matching annotations.

signal [1d numpy array, optional] The original signal of the two annotations. Only used for plotting.

comparitor [Comparitor object] Object containing parameters about the two sets of annotations

```
>>> import wfdb
>>> from wfdb import processing
```

```
>>> sig, fields = wfdb.rdsamp('sample-data/100', channels=[0])
>>> ann_ref = wfdb.rdann('sample-data/100','atr')
>>> xqrs = processing.XQRS(sig=sig[:,0], fs=fields['fs'])
>>> xqrs.detect()
```

```
>>> comparitor = processing.compare_annotations(ann_ref.sample[1:],
                                                xqrs.qrs_inds,
                                                int(0.1 * fields['fs']),
                                                sig[:,0])
>>> comparitor.print_summary()
>>> comparitor.plot()
```

wfdb.processing.**benchmark_mitdb**(*detector*, *verbose=False*)
Benchmark a qrs detector against mitdb's records.

detector [function] The detector function.

verbose [bool, optional] The verbose option of the detector function.

comparitors [list] List of Comparitor objects run on the records.

specificity [float] Aggregate specificity.

positive_predictivity [float] Aggregate positive_predictivity.

false_positive_rate [float] Aggregate false_positive_rate.

TODO: - remove non-qrs detections from reference annotations - allow kwargs

```
>>> import wfdb
>> from wfdb.processing import benchmark_mitdb, xqrs_detect
```

```
>>> comparitors, spec, pp, fpr = benchmark_mitdb(xqrs_detect)
```

CHAPTER 4

Core Components

A subset of the above components are accessible by directly importing the base package.

4.1 wfdb

These core components are accessible by importing the *wfdb* package directly, as well as from their respective sub-packages.

4.1.1 WFDB Records

wfdb.**rdrecord**(*record_name*, *sampfrom=0*, *sampto=None*, *channels=None*, *physical=True*, *pb_dir=None*, *m2s=True*, *smooth_frames=True*, *ignore_skew=False*, *return_res=64*, *force_channels=True*, *channel_names=None*, *warn_empty=False*)

Read a WFDB record and return the signal and record descriptors as attributes in a Record or MultiRecord object.

record_name [str] The name of the WFDB record to be read, without any file extensions. If the argument contains any path delimiter characters, the argument will be interpreted as PATH/BASE_RECORD. Both relative and absolute paths are accepted. If the *pb_dir* parameter is set, this parameter should contain just the base record name, and the files fill be searched for remotely. Otherwise, the data files will be searched for in the local path.

sampfrom [int, optional] The starting sample number to read for all channels.

sampto [int, or 'end', optional] The sample number at which to stop reading for all channels. Reads the entire duration by default.

channels [list, optional] List of integer indices specifying the channels to be read. Reads all channels by default.

physical [bool, optional] Specifies whether to return signals in physical units in the *p_signal* field (True), or digital units in the *d_signal* field (False).

pb_dir [str, optional] Option used to stream data from Physiobank. The Physiobank database directory from which to find the required record files. eg. For record '100' in 'http://physionet.org/physiobank/database/mitdb' pb_dir='mitdb'.

m2s [bool, optional] Used when reading multi-segment records. Specifies whether to directly return a wfdb MultiRecord object (False), or to convert it into and return a wfdb Record object (True).

smooth_frames [bool, optional] Used when reading records with signals having multiple samples per frame. Specifies whether to smooth the samples in signals with more than one sample per frame and return an (MxN) uniform numpy array as the *d_signal* or *p_signal* field (True), or to return a list of 1d numpy arrays containing every expanded sample as the *e_d_signal* or *e_p_signal* field (False).

ignore_skew [bool, optional] Used when reading records with at least one skewed signal. Specifies whether to apply the skew to align the signals in the output variable (False), or to ignore the skew field and load in all values contained in the dat files unaligned (True).

return_res [int, optional] The numpy array dtype of the returned signals. Options are: 64, 32, 16, and 8, where the value represents the numpy int or float dtype. Note that the value cannot be 8 when physical is True since there is no float8 format.

force_channels [bool, optional] Used when reading multi-segment variable layout records. Whether to update the layout specification record, and the converted Record object if *m2s* is True, to match the input *channels* argument, or to omit channels in which no read segment contains the signals.

channel_names [list, optional] List of channel names to return. If this parameter is specified, it takes precedence over *channels*.

warn_empty [bool, optional] Whether to display a warning if the specified channel indices or names are not contained in the record, and no signal is returned.

record [Record or MultiRecord] The wfdb Record or MultiRecord object representing the contents of the record read.

If a signal range or channel selection is specified when calling this function, the resulting attributes of the returned object will be set to reflect the section of the record that is actually read, rather than necessarily the entire record. For example, if *channels=[0, 1, 2]* is specified when reading a 12 channel record, the 'n_sig' attribute will be 3, not 12.

The *rdsamp* function exists as a simple alternative to *rdrecord* for the common purpose of extracting the physical signals and a few important descriptor fields.

```
>>> record = wfdb.rdrecord('sample-data/test01_00s', sampfrom=800,
                           channels=[1, 3])
```

wfdb.**rdsamp**(*record_name*, *sampfrom=0*, *sampto=None*, *channels=None*, *pb_dir=None*, *channel_names=None*, *warn_empty=False*)

Read a WFDB record, and return the physical signals and a few important descriptor fields.

record_name [str] The name of the WFDB record to be read (without any file extensions). If the argument contains any path delimiter characters, the argument will be interpreted as PATH/baserecord and the data files will be searched for in the local path.

sampfrom [int, optional] The starting sample number to read for all channels.

sampto [int, or 'end', optional] The sample number at which to stop reading for all channels. Reads the entire duration by default.

channels [list, optional] List of integer indices specifying the channels to be read. Reads all channels by default.

pb_dir [str, optional] Option used to stream data from Physiobank. The Physiobank database directory from which to find the required record files. eg. For record '100' in 'http://physionet.org/physiobank/database/mitdb' pb_dir='mitdb'.

channel_names [list, optional] List of channel names to return. If this parameter is specified, it takes precedence over *channels*.

warn_empty [bool, optional] Whether to display a warning if the specified channel indices or names are not contained in the record, and no signal is returned.

signals [numpy array] A 2d numpy array storing the physical signals from the record.

fields [dict] A dictionary containing several key attributes of the read record:

- fs: The sampling frequency of the record
- units: The units for each channel
- sig_name: The signal name for each channel
- comments: Any comments written in the header

If a signal range or channel selection is specified when calling this function, the resulting attributes of the returned object will be set to reflect the section of the record that is actually read, rather than necessarily the entire record. For example, if *channels=[0, 1, 2]* is specified when reading a 12 channel record, the 'n_sig' attribute will be 3, not 12.

The *rdrecord* function is the base function upon which this one is built. It returns all attributes present, along with the signals, as attributes in a *Record* object. The function, along with the returned data type, has more options than *rdsamp* for users who wish to more directly manipulate WFDB content.

```
>>> signals, fields = wfdb.rdsamp('sample-data/test01_00s',
                                  sampfrom=800,
                                  channel =[1,3])
```

wfdb.**wrsamp**(*record_name*, *fs*, *units*, *sig_name*, *p_signal=None*, *d_signal=None*, *fmt=None*, *adc_gain=None*, *baseline=None*, *comments=None*, *base_time=None*, *base_date=None*, *write_dir=''*)

Write a single segment WFDB record, creating a WFDB header file and any associated dat files.

record_name [str] The string name of the WFDB record to be written (without any file extensions).

fs [int, or float] The sampling frequency of the record.

units [list] A list of strings giving the units of each signal channel.

sig_name : A list of strings giving the signal name of each signal channel.

p_signal [numpy array, optional] An (MxN) 2d numpy array, where M is the signal length. Gives the physical signal values intended to be written. Either p_signal or d_signal must be set, but not both. If p_signal is set, this method will use it to perform analogue-digital conversion, writing the resultant digital values to the dat file(s). If fmt is set, gain and baseline must be set or unset together. If fmt is unset, gain and baseline must both be unset.

d_signal [numpy array, optional] An (MxN) 2d numpy array, where M is the signal length. Gives the digital signal values intended to be directly written to the dat file(s). The dtype must be an integer type. Either p_signal or d_signal must be set, but not both. In addition, if d_signal is set, fmt, gain and baseline must also all be set.

fmt [list, optional] A list of strings giving the WFDB format of each file used to store each channel. Accepted formats are: '80','212","16','24', and '32'. There are other WFDB formats as specified by: https://www.

physionet.org/physiotools/wag/signal-5.htm but this library will not write (though it will read) those file types.

adc_gain [list, optional] A list of numbers specifying the ADC gain.

baseline [list, optional] A list of integers specifying the digital baseline.

comments [list, optional] A list of string comments to be written to the header file.

base_time [str, optional] A string of the record's start time in 24h 'HH:MM:SS(.ms)' format.

base_date [str, optional] A string of the record's start date in 'DD/MM/YYYY' format.

write_dir [str, optional] The directory in which to write the files.

This is a gateway function, written as a simple method to write WFDB record files using the most common parameters. Therefore not all WFDB fields can be set via this function.

For more control over attributes, create a *Record* object, manually set its attributes, and call its *wrsamp* instance method. If you choose this more advanced method, see also the *set_defaults*, *set_d_features*, and *set_p_features* instance methods to help populate attributes.

```
>>> # Read part of a record from Physiobank
>>> signals, fields = wfdb.rdsamp('a103l', sampfrom=50000, channels=[0,1],
                                  pb_dir='challenge/2015/training')
>>> # Write a local WFDB record (manually inserting fields)
>>> wfdb.wrsamp('ecgrecord', fs = 250, units=['mV', 'mV'],
                sig_name=['I', 'II'], p_signal=signals, fmt=['16', '16'])
```

class wfdb.**Record**(*p_signal=None*, *d_signal=None*, *e_p_signal=None*, *e_d_signal=None*, *record_name=None*, *n_sig=None*, *fs=None*, *counter_freq=None*, *base_counter=None*, *sig_len=None*, *base_time=None*, *base_date=None*, *file_name=None*, *fmt=None*, *samps_per_frame=None*, *skew=None*, *byte_offset=None*, *adc_gain=None*, *baseline=None*, *units=None*, *adc_res=None*, *adc_zero=None*, *init_value=None*, *checksum=None*, *block_size=None*, *sig_name=None*, *comments=None*)

The class representing single segment WFDB records.

Record objects can be created using the initializer, by reading a WFDB header with *rdheader*, or a WFDB record (header and associated dat files) with *rdrecord*.

The attributes of the Record object give information about the record as specified by: https://www.physionet.org/physiotools/wag/header-5.htm

In addition, the d_signal and p_signal attributes store the digital and physical signals of WFDB records with at least one channel.

```
>>> record = wfdb.Record(record_name='r1', fs=250, n_sig=2, sig_len=1000,
                         file_name=['r1.dat','r1.dat'])
```

adc(*expanded=False*, *inplace=False*)

Performs analogue to digital conversion of the physical signal stored in p_signal if expanded is False, or e_p_signal if expanded is True.

The p_signal/e_p_signal, fmt, gain, and baseline fields must all be valid.

If inplace is True, the adc will be performed inplace on the variable, the d_signal/e_d_signal attribute will be set, and the p_signal/e_p_signal field will be set to None.

expanded [bool, optional] Whether to transform the *e_p_signal* attribute (True) or the *p_signal* attribute (False).

inplace [bool, optional] Whether to automatically set the object's corresponding digital signal attribute and set the physical signal attribute to None (True), or to return the converted signal as a separate variable without changing the original physical signal attribute (False).

d_signal [numpy array, optional] The digital conversion of the signal. Either a 2d numpy array or a list of 1d numpy arrays.

```
>>> import wfdb
>>> record = wfdb.rdsamp('sample-data/100')
>>> d_signal = record.adc()
>>> record.adc(inplace=True)
>>> record.dac(inplace=True)
```

dac(*expanded=False*, *return_res=64*, *inplace=False*)

Performs the digital to analogue conversion of the signal stored in *d_signal* if expanded is False, or *e_d_signal* if expanded is True.

The d_signal/e_d_signal, fmt, gain, and baseline fields must all be valid.

If inplace is True, the dac will be performed inplace on the variable, the p_signal/e_p_signal attribute will be set, and the d_signal/e_d_signal field will be set to None.

expanded [bool, optional] Whether to transform the *e_d_signal attribute* (True) or the *d_signal* attribute (False).

inplace [bool, optional] Whether to automatically set the object's corresponding physical signal attribute and set the digital signal attribute to None (True), or to return the converted signal as a separate variable without changing the original digital signal attribute (False).

p_signal [numpy array, optional] The physical conversion of the signal. Either a 2d numpy array or a list of 1d numpy arrays.

```
>>> import wfdb
>>> record = wfdb.rdsamp('sample-data/100', physical=False)
>>> p_signal = record.dac()
>>> record.dac(inplace=True)
>>> record.adc(inplace=True)
```

wrsamp(*expanded=False*, *write_dir=''*)

Write a wfdb header file and any associated dat files from this object.

expanded [bool, optional] Whether to write the expanded signal (e_d_signal) instead of the uniform signal (d_signal).

write_dir [str, optional] The directory in which to write the files.

class wfdb.**MultiRecord**(*segments=None*, *layout=None*, *record_name=None*, *n_sig=None*, *fs=None*, *counter_freq=None*, *base_counter=None*, *sig_len=None*, *base_time=None*, *base_date=None*, *seg_name=None*, *seg_len=None*, *comments=None*, *sig_name=None*, *sig_segments=None*)

The class representing multi-segment WFDB records.

MultiRecord objects can be created using the initializer, or by reading a multi-segment WFDB record using 'rdrecord' with the *m2s* (multi to single) input parameter set to False.

The attributes of the MultiRecord object give information about the entire record as specified by: https://www.physionet.org/physiotools/wag/header-5.htm

In addition, the *segments* parameter is a list of Record objects representing each individual segment, or None representing empty segments, of the entire multi-segment record.

Notably, this class has no attribute representing the signals as a whole. The 'multi_to_single' instance method can be called on MultiRecord objects to return a single segment representation of the record as a Record object. The resulting Record object will have its 'p_signal' field set.

```
>>> record_m = wfdb.MultiRecord(record_name='rm', fs=50, n_sig=8,
                                sig_len=9999, seg_name=['rm_1', '~', rm_2'],
                                seg_len=[800, 200, 900])
>>> # Get a MultiRecord object
>>> record_s = wfdb.rdsamp('s00001-2896-10-10-00-31', m2s=False)
>>> # Turn it into a
>>> record_s = record_s.multi_to_single()
```

record_s initially stores a *MultiRecord* object, and is then converted into a *Record* object.

multi_to_single(*physical*, *return_res=64*)

Create a Record object from the MultiRecord object. All signal segments will be combined into the new object's *p_signal* or *d_signal* field. For digital format, the signals must have the same storage format, baseline, and adc_gain in all segments.

physical [bool] Whether to convert the physical or digital signal.

return_res [int, optional] The return resolution of the *p_signal* field. Options are: 64, 32, and 16.

record [wfdb Record] The single segment record created.

4.1.2 WFDB Anotations

wfdb.**rdann**(*record_name, extension, sampfrom=0, sampto=None, shift_samps=False, pb_dir=None, return_label_elements=['symbol'], summarize_labels=False*)

Read a WFDB annotation file record_name.extension and return an Annotation object.

record_name [str] The record name of the WFDB annotation file. ie. for file '100.atr', record_name='100'.

extension [str] The annotatator extension of the annotation file. ie. for file '100.atr', extension='atr'.

sampfrom [int, optional] The minimum sample number for annotations to be returned.

sampto [int, optional] The maximum sample number for annotations to be returned.

shift_samps [bool, optional] Specifies whether to return the sample indices relative to *sampfrom* (True), or sample 0 (False).

pb_dir [str, optional] Option used to stream data from Physiobank. The Physiobank database directory from which to find the required annotation file. eg. For record '100' in 'http://physionet.org/physiobank/database/mitdb': pb_dir='mitdb'.

return_label_elements [list, optional] The label elements that are to be returned from reading the annotation file. A list with at least one of the following options: 'symbol', 'label_store', 'description'.

summarize_labels [bool, optional] If True, assign a summary table of the set of annotation labels contained in the file to the 'contained_labels' attribute of the returned object. This table will contain the columns: ['label_store', 'symbol', 'description', 'n_occurences']

annotation [Annotation] The Annotation object. Call help(wfdb.Annotation) for the attribute descriptions.

For every annotation sample, the annotation file explictly stores the 'sample' and 'symbol' fields, but not necessarily the others. When reading annotation files using this function, fields which are not stored in the file will either take their default values of 0 or None, or will be carried over from their previous values if any.

```
>>> ann = wfdb.rdann('sample-data/100', 'atr', sampto=300000)
```

wfdb.**wrann**(*record_name*, *extension*, *sample*, *symbol=None*, *subtype=None*, *chan=None*, *num=None*, *aux_note=None*, *label_store=None*, *fs=None*, *custom_labels=None*, *write_dir=''*)

Write a WFDB annotation file.

Specify at least the following:

- The record name of the WFDB record (record_name)
- The annotation file extension (extension)
- The annotation locations in samples relative to the beginning of the record (sample)
- Either the numerical values used to store the labels (*label_store*), or more commonly, the display symbols of each label (*symbol*).

record_name [str] The string name of the WFDB record to be written (without any file extensions).

extension [str] The string annotation file extension.

sample [numpy array] A numpy array containing the annotation locations in samples relative to the beginning of the record.

symbol [list, or numpy array, optional] The symbols used to display the annotation labels. List or numpy array. If this field is present, *label_store* must not be present.

subtype [numpy array, optional] A numpy array containing the marked class/category of each annotation.

chan [numpy array, optional] A numpy array containing the signal channel associated with each annotation.

num [numpy array, optional] A numpy array containing the labelled annotation number for each annotation.

aux_note [list, optional] A list containing the auxiliary information string (or None for annotations without notes) for each annotation.

label_store [numpy array, optional] A numpy array containing the integer values used to store the annotation labels. If this field is present, *symbol* must not be present.

fs [int, or float, optional] The numerical sampling frequency of the record to be written to the file.

custom_labels [pandas dataframe, optional] The map of custom defined annotation labels used for this annotation, in addition to the standard WFDB annotation labels. Custom labels are defined by two or three fields:

- The integer values used to store custom annotation labels in the file (optional)
- Their short display symbols
- Their long descriptions.

This input argument may come in four formats:

1. A pandas.DataFrame object with columns: ['label_store', 'symbol', 'description']
2. A pandas.DataFrame object with columns: ['symbol', 'description'] If this option is chosen, label_store values are automatically chosen.
3. A list or tuple of tuple triplets, with triplet elements representing: (label_store, symbol, description).
4. A list or tuple of tuple pairs, with pair elements representing: (symbol, description). If this option is chosen, label_store values are automatically chosen.

If the *label_store* field is given for this function, and *custom_labels* is defined, *custom_labels* must contain *label_store* in its mapping. ie. it must come in format 1 or 3 above.

write_dir [str, optional] The directory in which to write the annotation file

This is a gateway function, written as a simple way to write WFDB annotation files without needing to explicity create an Annotation object. You may also create an Annotation object, manually set its attributes, and call its *wrann* instance method.

Each annotation stored in a WFDB annotation file contains a sample field and a label field. All other fields may or may not be present.

```
>>> # Read an annotation as an Annotation object
>>> annotation = wfdb.rdann('b001', 'atr', pb_dir='cebsdb')
>>> # Write a copy of the annotation file
>>> wfdb.wrann('b001', 'cpy', annotation.sample, annotation.symbol)
```

wfdb.**show_ann_labels**()
Display the standard wfdb annotation label mapping.

```
>>> show_ann_labels()
```

wfdb.**show_ann_classes**()
Display the standard wfdb annotation classes

```
>>> show_ann_classes()
```

class wfdb.**Annotation**(*record_name*, *extension*, *sample*, *symbol=None*, *subtype=None*, *chan=None*, *num=None*, *aux_note=None*, *fs=None*, *label_store=None*, *description=None*, *custom_labels=None*, *contained_labels=None*)
The class representing WFDB annotations.

Annotation objects can be created using the initializer, or by reading a WFDB annotation file with *rdann*.

The attributes of the Annotation object give information about the annotation as specified by: https://www.physionet.org/physiotools/wag/annot-5.htm

Call *show_ann_labels()* to see the list of standard annotation codes. Any text used to label annotations that are not one of these codes should go in the 'aux_note' field rather than the 'sym' field.

```
>>> ann1 = wfdb.Annotation(record_name='rec1', extension='atr',
                           sample=[10,20,400], symbol=['N','N','['],
                           aux_note=[None, None, 'Serious Vfib'])
```

wrann(*write_fs=False*, *write_dir=''*)
Write a WFDB annotation file from this object.

write_fs [bool, optional] Whether to write the *fs* attribute to the file.

4.1.3 Downloading

wfdb.**get_dbs**()
Get a list of all the Physiobank databases available.

```
>>> dbs = get_dbs()
```

wfdb.**get_record_list**(*db_dir*, *records='all'*)
Get a list of records belonging to a database.

db_dir [str] The database directory, usually the same as the database slug. The location to look for a RECORDS file.

records [list, optional] A Option used when this function acts as a helper function. Leave as default 'all' to get all records.

```
>>> wfdb.get_record_list('mitdb')
```

wfdb.**dl_database**(*db_dir*, *dl_dir*, *records='all'*, *annotators='all'*, *keep_subdirs=True*, *overwrite=False*)

Download WFDB record (and optionally annotation) files from a Physiobank database. The database must contain a 'RECORDS' file in its base directory which lists its WFDB records.

db_dir [str] The Physiobank database directory to download. eg. For database: 'http://physionet.org/physiobank/database/mitdb', db_dir='mitdb'.

dl_dir [str] The full local directory path in which to download the files.

records [list, or 'all', optional] A list of strings specifying the WFDB records to download. Leave as 'all' to download all records listed in the database's RECORDS file. eg. records=['test01_00s', test02_45s] for database: https://physionet.org/physiobank/database/macecgdb/

annotators [list, 'all', or None, optional] A list of strings specifying the WFDB annotation file types to download along with the record files. Is either None to skip downloading any annotations, 'all' to download all annotation types as specified by the ANNOTATORS file, or a list of strings which each specify an annotation extension. eg. annotators = ['anI'] for database: https://physionet.org/physiobank/database/prcp/

keep_subdirs [bool, optional] Whether to keep the relative subdirectories of downloaded files as they are organized in Physiobank (True), or to download all files into the same base directory (False).

overwrite [bool, optional] If True, all files will be redownloaded regardless. If False, existing files with the same name and relative subdirectory will be checked. If the local file is the same size as the online file, the download is skipped. If the local file is larger, it will be deleted and the file will be redownloaded. If the local file is smaller, the file will be assumed to be partially downloaded and the remaining bytes will be downloaded and appended.

```
>>> wfdb.dl_database('ahadb', os.getcwd())
```

wfdb.**dl_files**(*db*, *dl_dir*, *files*, *keep_subdirs=True*, *overwrite=False*)

Download specified files from a Physiobank database.

db [str] The Physiobank database directory to download. eg. For database: 'http://physionet.org/physiobank/database/mitdb', db='mitdb'.

dl_dir [str] The full local directory path in which to download the files.

files [list] A list of strings specifying the file names to download relative to the database base directory.

keep_subdirs [bool, optional] Whether to keep the relative subdirectories of downloaded files as they are organized in Physiobank (True), or to download all files into the same base directory (False).

overwrite [bool, optional] If True, all files will be redownloaded regardless. If False, existing files with the same name and relative subdirectory will be checked. If the local file is the same size as the online file, the download is skipped. If the local file is larger, it will be deleted and the file will be redownloaded. If the local file is smaller, the file will be assumed to be partially downloaded and the remaining bytes will be downloaded and appended.

```
>>> wfdb.dl_files('ahadb', os.getcwd(),
                  ['STAFF-Studies-bibliography-2016.pdf', 'data/001a.hea',
                  'data/001a.dat'])
```

wfdb.**set_db_index_url**(*db_index_url='http://physionet.org/physiobank/database/'*)
Set the database index url to a custom value, to stream remote files from another location.

db_index_url [str, optional] The desired new database index url. Leave as default to reset to the physiobank index url.

4.1.4 Plotting

wfdb.**plot_items**(*signal=None, ann_samp=None, ann_sym=None, fs=None, time_units='samples', sig_name=None, sig_units=None, ylabel=None, title=None, sig_style=[''], ann_style=['r*'], ecg_grids=[], figsize=None, return_fig=False*)
Subplot individual channels of signals and/or annotations.

signal [1d or 2d numpy array, optional] The uniformly sampled signal to be plotted. If signal.ndim is 1, it is assumed to be a one channel signal. If it is 2, axes 0 and 1, must represent time and channel number respectively.

ann_samp: list, optional A list of annotation locations to plot, with each list item corresponding to a different channel. List items may be:

- 1d numpy array, with values representing sample indices. Empty arrays are skipped.
- list, with values representing sample indices. Empty lists are skipped.
- None. For channels in which nothing is to be plotted.

If *signal* is defined, the annotation locations will be overlaid on the signals, with the list index corresponding to the signal channel. The length of *annotation* does not have to match the number of channels of *signal*.

ann_sym: list, optional A list of annotation symbols to plot, with each list item corresponding to a different channel. List items should be lists of strings. The symbols are plotted over the corresponding *ann_samp* index locations.

fs [int or float, optional] The sampling frequency of the signals and/or annotations. Used to calculate time intervals if *time_units* is not 'samples'. Also required for plotting ecg grids.

time_units [str, optional] The x axis unit. Allowed options are: 'samples', 'seconds', 'minutes', and 'hours'.

sig_name [list, optional] A list of strings specifying the signal names. Used with *sig_units* to form y labels, if *ylabel* is not set.

sig_units [list, optional] A list of strings specifying the units of each signal channel. Used with *sig_name* to form y labels, if *ylabel* is not set. This parameter is required for plotting ecg grids.

ylabel [list, optional] A list of strings specifying the final y labels. If this option is present, *sig_name* and *sig_units* will not be used for labels.

title [str, optional] The title of the graph.

sig_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each signal channel. The list length should match the number of signal channels. If the list has a length of 1, the style will be used for all channels.

ann_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each annotation channel. If the list has a length of 1, the style will be used for all channels.

ecg_grids [list, optional] A list of integers specifying channels in which to plot ecg grids. May also be set to 'all' for all channels. Major grids at 0.5mV, and minor grids at 0.125mV. All channels to be plotted with grids must have *sig_units* equal to 'uV', 'mV', or 'V'.

figsize [tuple, optional] Tuple pair specifying the width, and height of the figure. It is the 'figsize' argument passed into matplotlib.pyplot's *figure* function.

return_fig [bool, optional] Whether the figure is to be returned as an output argument.

figure [matplotlib figure, optional] The matplotlib figure generated. Only returned if the 'return_fig' parameter is set to True.

```
>>> record = wfdb.rdrecord('sample-data/100', sampto=3000)
>>> ann = wfdb.rdann('sample-data/100', 'atr', sampto=3000)
```

```
>>> wfdb.plot_items(signal=record.p_signal,
                    annotation=[ann.sample, ann.sample],
                    title='MIT-BIH Record 100', time_units='seconds',
                    figsize=(10,4), ecg_grids='all')
```

wfdb.**plot_wfdb**(*record=None, annotation=None, plot_sym=False, time_units='samples', title=None, sig_style=[''], ann_style=['r*'], ecg_grids=[], figsize=None, return_fig=False*)

Subplot individual channels of a wfdb record and/or annotation.

This function implements the base functionality of the *plot_items* function, while allowing direct input of wfdb objects.

If the record object is input, the function will extract from it:

- signal values, from the *p_signal* (priority) or *d_signal* attribute
- sampling frequency, from the *fs* attribute
- signal names, from the *sig_name* attribute
- signal units, from the *units* attribute

If the annotation object is input, the function will extract from it:

- sample locations, from the *sample* attribute
- symbols, from the *symbol* attribute
- the annotation channels, from the *chan* attribute
- the sampling frequency, from the *fs* attribute if present, and if fs was not already extracted from the *record* argument.

record [wfdb Record, optional] The Record object to be plotted

annotation [wfdb Annotation, optional] The Annotation object to be plotted

plot_sym [bool, optional] Whether to plot the annotation symbols on the graph.

time_units [str, optional] The x axis unit. Allowed options are: 'samples', 'seconds', 'minutes', and 'hours'.

title [str, optional] The title of the graph.

sig_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each signal channel. The list length should match the number of signal channels. If the list has a length of 1, the style will be used for all channels.

ann_style [list, optional] A list of strings, specifying the style of the matplotlib plot for each annotation channel. The list length should match the number of annotation channels. If the list has a length of 1, the style will be used for all channels.

ecg_grids [list, optional] A list of integers specifying channels in which to plot ecg grids. May also be set to 'all' for all channels. Major grids at 0.5mV, and minor grids at 0.125mV. All channels to be plotted with grids must have *sig_units* equal to 'uV', 'mV', or 'V'.

figsize [tuple, optional] Tuple pair specifying the width, and height of the figure. It is the 'figsize' argument passed into matplotlib.pyplot's *figure* function.

return_fig [bool, optional] Whether the figure is to be returned as an output argument.

figure [matplotlib figure, optional] The matplotlib figure generated. Only returned if the 'return_fig' option is set to True.

```
>>> record = wfdb.rdrecord('sample-data/100', sampto=3000)
>>> annotation = wfdb.rdann('sample-data/100', 'atr', sampto=3000)
```

```
>>> wfdb.plot_wfdb(record=record, annotation=annotation, plot_sym=True
                   time_units='seconds', title='MIT-BIH Record 100',
                   figsize=(10,4), ecg_grids='all')
```

wfdb.**plot_all_records**(*directory=''*)

Plot all wfdb records in a directory (by finding header files), one at a time, until the 'enter' key is pressed.

directory [str, optional] The directory in which to search for WFDB records. Defaults to current working directory.

CHAPTER 5

Other Content

5.1 Installation

The distribution is hosted on pypi at: https://pypi.python.org/pypi/wfdb/. To directly install the package from pypi without needing to explicitly download content, run from your terminal:

```
$ pip install wfdb
```

The development version is hosted at: https://github.com/MIT-LCP/wfdb-python. This repository also contains demo scripts and example data. To install the development version, clone or download the repository, navigate to the base directory, and run:

```
$ pip install .
```

5.2 WFDB Specifications

The wfdb-python package is built according to the specifications of the original WFDB package.

- WFDB Software Package
- WFDB Applications Guide
- WFDB Header Specifications
- WFDB Signal Specifications
- WFDB Annotation Specifications

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

CHAPTER 7

Authors

Chen Xie

Julien Dubiel

Python Module Index

W

Index

S

W

X