WFDB_tools
A MATLAB interface to the WFDB library

Jonas Carlson

November 30, 2003

The most recent versions of the software described here may be freely downloaded from PhysioNet (`http://www.physionet.org/`). An HTML version of this guide is available at `http://www.physionet.org/physiotools/matlab/-wfdb_tools/`.

Contents

Preface

The Waveform Database interface library (the WFDB library) is a package of C-callable functions that provide clean and uniform access to digitized, annotated signals stored in a variety of formats. These functions, although originally designed for use with databases of electrocardiograms, are useful for dealing with any similar collection of digitized signals, which may or may not be annotated. The WFDB library has evolved to support the development of numerous databases that include signals such as blood pressure, respiration, oxygen saturation, EEG, as well as ECGs. Thus the WFDB library is considerably more than an ECG database interface.

This guide documents WFDB_tools, a set of MATLAB functions that enables the MATLAB user to take full advantage of the WFDB library and explore or create databases containing a wide variety of signals.

The WFDB_tools functions are not self-contained; rather, they are 'wrappers' for the WFDB library functions (i.e. the WFDB library must be installed for the MATLAB functions to work). The wrappers work much in the same way as the WFDB library functions, and the effort is to keep them as true to their counterparts as possible. The usual way to work in MATLAB is to get all results from one function, and therefore the wrappers may seem 'low-level' in comparison. Combining just a few of the wrappers in an m-file, however, would produce the 'high-level' way of working in MATLAB while keeping the full control of data handling that can only be obtained from the 'low-level' C-like wrappers.

This guide includes several short tutorial examples that illustrate how to read and write signals and annotations using these wrappers. Eventually, it will also contain descriptions of all the wrapper functions available to the MATLAB user. The current version of the guide does not include descriptions of all of the functions. Note, however, that MATLAB help files for all of the wrappers are included in the WFDB_tools package, so it is always possible within MATLAB to use a command such as

```
help WFDB_sampfreq
```

to obtain information about how to use any of these wrappers, including those that are not yet documented in this guide.

The set of wrappers is nearly complete. WFDB library functions for which no wrappers currently exist are wfdbinit, ungetann, sample, sample_valid, se-

tannstr, setanndesc, setecgstr, calopen, getcal, putcal, newcal, flushcal, setmsheader, setwfdb, setibsize, and setobsize.

Chapter 1

Installing the WFDB_tools wrappers

Important: the WFDB_tools wrappers have been developed and tested with MATLAB *R13.* It is highly unlikely that they can be made to work with an older version of MATLAB. Initial development and testing was under Red Hat Linux 8.0, using WFDB library version 10.3.2; the wrappers have also been tested under Red Hat Linux 9.0 and WFDB library version 10.3.11. More recently, they have also been tested under MS-Windows XP, also with WFDB library version 10.3.11. It may be possible to recompile these wrappers and to use them with MATLAB R13 on other platforms such as Mac OS/X or Solaris, but doing so is currently unsupported.

1.1 Preparation

Before installing the WFDB_tools wrappers, install the WFDB Software Package (`http://www.physionet.org/physiotools/wfdb.shtml`), and verify that it is working properly on your system.

Download the WFDB_tools package (`http://www.physionet.org/physiotools/matlab/WFDB_tools.tar.gz`) and unpack it using a command such as

```
tar xfvz WFDB_tools.tar.gz
```

This creates a directory named `WFDB_tools`, which contains the wrapper source files (`src/*.c`), help files (`help/*.m`), precompiled binary files (`linux/*.mexglx`, for x86 GNU/Linux, and `windows/*.dll`, for MS-Windows), documentation (in `doc/`), and tutorial examples (in `examples/`).

Follow the instructions given in the `00README` file (in the top-level `WFDB_tools` directory) to install the WFDB_tools on your system.

All functions have explanatory m-files available through MATLAB's `help` function. In the case of toolbox installation (described below) a list of all functions is found using `help WFDB_tools`.

1.2 Using as a directory

The directory `WFDB_tools` can be put anywhere. All MATLAB programs must then use the command `addpath` to make the directory available (it is probably not a good idea to put other program files inside the `WFDB_tools` directory).

1.3 Using as a MATLAB toolbox

With sufficient write privileges, it should be possible to install the WFDB tools as a MATLAB toolbox. It should then be put in the directory:

matlabroot/toolbox/WFDB_tools

To make it available to MATLAB, this directory must be added to `pathdef.m` (in the `toolbox/local` directory) and, if the toolbox path cache is enabled, the command `rehash toolboxcache` should be issued when MATLAB is started.

1.4 The WFDB library and applications

The *WFDB Programmer's Guide* (`http://www.physionet.org/physiotools/wpg/`), which documents the C-language WFDB library, is recommended as a source of additional information and examples. The *WFDB Applications Guide* (`http://www.physionet.org/physiotools/wag/`) describes many stand-alone programs that use the WFDB library to read and write digitized signals and annotations. If the WFDB Software Package has been correctly installed, you can run these programs from a terminal window or from within MATLAB to perform a wide variety of signal processing and analysis tasks.

1.5 Sample, signal, and annotator numbers

Several WFDB library functions, and most of the stand-alone WFDB applications, accept arguments that specify a specific sample within a digitized signal (a *sample number*, a specific signal within a set of signals (a *signal number*, or a specific set of annotations (an *annotator number*). The first sample number in a signal has sample number 0, not 1; similarly, the first signal has signal number 0, and the first annotator has annotator number 0. The WFDB_tools functions use the same zero-based sample, signal, and annotator numbers as the WFDB library functions that they wrap. This point is a possible source of confusion if you become accustomed to thinking of these numbers as array indices (which, in C, is exactly what they are); it may be best to think of them simply as identification numbers for the objects with which they are associated.

Chapter 2

Using the WFDB_tools library

Using simple examples, this chapter illustrates how to use the WFDB_tools wrappers to read and write signals and annotations. Additional information about the wrappers in these examples, and about the other wrappers in the library, can be found in the next chapter.

2.1 Reading signals

Assuming that the WFDB Software Package has been installed correctly, the record "100s" should be available. Reading the first ten samples of this record using MATLAB would be done as:

```
>> S = WFDB_isigopen('100s')
S =
2x1 struct array with fields:
    fname
    desc
    units
    gain
    initval
    group
    fmt
    spf
    bsize
    adcres
    adczero
    baseline
    nsamp
    cksum
```

```
>> DATA = WFDB_getvec(length(S), 10)
DATA =
         995        1011
         995        1011
         995        1011
         995        1011
         995        1011
         995        1011
         995        1011
         995        1011
        1000        1008
         997        1008
```

The first command, `S = WFDB_isigopen('100s')`, reads the header file of record 100s and returns the information in a structure (`S`, in this case). The length of `S` equals the number of signals in the data file. The fields of `S` contain the signal settings. To access, for example, the gain of signal 0 and the description of signal 1, use the commands:

```
>> S(1).gain
ans =
   200
>> S(2).desc
ans =
V5
```

(Remember: the first signal is signal 0, not signal 1! Its attributes are found in the first structure, `S(1)`. This will matter in later examples.)

The second command, `DATA = WFDB_getvec(length(S), 10)`, reads data from the previously opened record. The two input parameters are the number of signals (found as `length(S)`) and the desired number of samples (if omitted, the whole record is read).

Finally, don't forget

```
>> WFDB_wfdbquit
```

to close all open files.

An elaborated version of this example is provided in the `examples` directory of the WFDB_tools package (look for `example1.m`).

2.2 Reading annotations

This example illustrates how to open an annotation file, how to read annotations from it, and how to translate them into their mnemonic and description strings.

First, we need to create an 'Anninfo' structure containing the annotator name and mode of the annotation file:

```
>> A = WFDB_Anninfo(1)
A =
    name: 'a1'
    stat: 'WFDB_READ'
```

This might seem like a complicated way to go, but it reflects the way the underlying WFDB library works. The record 100s has an annotator named `atr`, so we need to change the `name` field of `A` before issuing the command to open the file.

```
>> A.name = 'atr'
A =
    name: 'atr'
    stat: 'WFDB_READ'
>> WFDB_annopen('100s', A)
```

Now that the annotation file is open, we may read the first two annotations, and take a closer look at the second one.

```
>> ANNOTATION = WFDB_getann(0, 2)
ANNOTATION =
2x1 struct array with fields:
    time
    anntyp
    subtyp
    chan
    num
    aux
>> ANNOTATION(2)
ans =
      time: 77
    anntyp: 1
    subtyp: 0
      chan: 0
       num: 0
       aux: ''
```

The first argument of WFDB_getann is the input annotator number. Since we are reading only one annotation file, its annotator number is 0. (If we had opened a second annotation file for reading, its annotator number would be 1, etc.)
Next, let's see what the annotation type (the `anntyp` field) means, in mnemonic and description:

```
>> WFDB_annstr(ANNOTATION(2).anntyp)
ans =
N
```

```
>> WFDB_anndesc(ANNOTATION(2).anntyp)
ans =
Normal beat
```

Finally,

```
>> WFDB_wfdbquit
```

An elaborated version of this example is provided in the `examples` directory of the WFDB_tools package (look for `example2.m`).

2.3 Creating an annotation file

In this example, we will create an annotation file that annotates the first two P-waves of record 100s. These are located at (roughly) sample numbers 315 and 610. First, let's find the annotation code for a P-wave:

```
>> WFDB_strann('p')
ans =
    24
```

The description is:

```
>> WFDB_anndesc(24)
ans =
P-wave peak
```

Create the two annotations:

```
>> ANN = WFDB_Annotation(2)
ANN =
1x2 struct array with fields:
    time
    anntyp
    subtyp
    chan
    num
    aux
>> ANN(1).time = 315;
>> ANN(1).anntyp = 24;
>> ANN(1).aux = 'First P-wave';
>> ANN(2).time = 610;
>> ANN(2).anntyp = 24;
>> ANN(2).aux = 'Second P-wave';
```

Now create an annotator structure:

```
>> A = WFDB_Anninfo(1)
A =
    name: 'a1'
    stat: 'WFDB_READ'
>> A.name = 'p';
>> A.stat = 'WFDB_WRITE';
```

Create an empty annotation file to hold the annotations:

```
>> WFDB_annopen('100s', A)
```

Write the annotations:

```
>> WFDB_putann(0, ANN)
```

The first argument of WFDB_putann is the output annotator number. Since we are writing only one annotation file, its annotator number is 0.
Close all open files:

```
>> WFDB_wfdbquit
```

The new annotation file, `100s.p`, will be located in MATLAB's current directory. The result may be verified using WAVE (under Linux) or GTKWave (under MS-Windows). These interactive viewers can be called from MATLAB using:

```
>> !wave -r 100s -a p &
```

(substitute `gtkwave` for `wave` under MS-Windows).
(Remember the !-sign to call system functions from MATLAB and the &-sign to run WAVE as a background process; depending on your setup, using & may cause an error, however, and you may need to run WAVE as a foreground process without the final '&' in the command.)

An elaborated version of this example is provided in the `examples` directory of the WFDB_tools package (look for `example3.m`).

2.4 Creating a signal file

Creating an output signal file is made in three steps: create a signal information structure, write the output signal data, and create the header file.

Assume we have data from three signals. We need to create a signal information structure using:

```
>> S = WFDB_Siginfo(3)
S =
1x3 struct array with fields:
    fname
    desc
```

```
units
gain
initval
group
fmt
spf
bsize
adcres
adczero
baseline
```

Now these fields need to be filled with appropriate values. All of them have default values to avoid producing errors, but it is unlikely that they will fit our signals. For example, if signal 0 is the X-lead of a Frank-lead ECG, we may want its description to be:

```
>> S(1).desc = 'Frank X';
```

and so on for all other fields. (Remember: the first signal is signal 0; its attributes are in `S(1)`.) When done, we create an empty signal file in which to write the data, using

```
>> WFDB_osigfopen(S)
```

We also need to supply the sampling frequency, for example 1 kHz:

```
>> WFDB_setsampfreq(1000);
```

and the basetime of the recording (i.e. the time of sample number 0). Assuming the recording was started when my oldest daughter was born:

```
>> WFDB_setbasetime('02:19:00 02/09/1999')
```

(Dates used by WFDB_tools are always in DD/MM/YYYY format; 02/09/1999 is 2 September, not February 9.)

Now we're done with providing signal information and it is time to write the actual signal data. This must be stored column-wise in a matrix (one signal per column, one sample per row). If our data is stored in the variable `DATA` we would use:

```
>> WFDB_putvec(DATA)
```

Finally, we need to record the information from `S` into a header (`.hea`) file for later use. We need to choose a name for the new record we are creating (avoiding the names of any existing records that we wish to read in the future); if we choose `test1` as the record name, we can create the header file by:

```
>> WFDB_newheader('test1')
```

These operations will create a header file, `test1.hea`, in the current directory, and a signal file with the name previously specified in the `fname` fields of the signal information structure, `S`. (Notice that `WFDB_newheader` must always be invoked *after* all of the samples have been written using `WFDB_putvec`, because the header file includes the number of samples in the record and checksums for each signal, which are not known by the WFDB library until all of the samples have been written.) As always, we use

```
>> WFDB_wfdbquit
```

to flush all pending output to the files, to close them, and to reset the internal WFDB library variables.

As in the previous example, we can inspect our results using WAVE or GTKWave:

```
>> !wave -r test1 &
```

An elaborated version of this example is provided in the `examples` directory of the WFDB_tools package (look for `example4.m`).

Chapter 3

WFDB_tools library functions

3.1 Selecting database records

These functions are used to open input and output signal and annotation files.

WFDB_annopen Opening input and output annotation files.

WFDB_isigopen Opening input signal files.

WFDB_osigopen Opening output signal files.

WFDB_osigfopen Opening output signal files by name.

3.1.1 WFDB_annopen

```
WFDB_annopen(RECORD, ANNINFO)
```

`RECORD`: Record name. String.
`ANNINFO`: WFDB_Anninfo structure(s).

This function opens input and output annotation files for a selected record. If RECORD begins with '+', previously opened annotation files are left open, and the record name is taken to be the remainder of RECORD after discarding the '+'. Otherwise, WFDB_annopen closes any previously opened annotation files, and takes all of RECORD as the record name. ANNINFO is a structure array created by WFDB_Anninfo (see section 3.7.1), with one array element for each annotator to be opened. The caller must fill in the WFDB_Anninfo structure array to specify the names of the annotators, and to indicate which annotators are to be read, and which are to be written. Input and output annotators may be listed in any order in ANNINFO. Annotator numbers (for both input and

output annotators) are assigned in the order in which the annotators appear in ANNINFO (the first annotator is number 0). For example, these instructions

```
>> a = WFDB_Anninfo(3);
>> a(1).name = 'a'; a(1).stat = 'WFDB_READ';
>> a(2).name = 'b'; a(2).stat = 'WFDB_WRITE';
>> a(3).name = 'c'; a(3).stat = 'WFDB_READ';
>> WFDB_annopen('100s', a)
```

attempt to open three annotation files for record '100s'. Annotator 'a' becomes input annotator 0, 'b' becomes output annotator 0, and 'c' becomes input annotator 1. Thus WFDB_getann(1) (see section 3.3.3) will read all annotations from annotator 'c', and WFDB_putann(0, ANN) (see section 3.3.4) will write an annotation for annotator 'b'. Input annotation files will be found if they are located in any of the directories in the WFDB path (see section 3.6.10). Output annotators are created in the current directory (but note that, under Unix at least, it is possible to specify annotator names such as '/here' or 'zzz/there' or even '../somewhere/else')

See also: WFDB_Anninfo (3.7.1), WFDB_getann (3.3.3), WFDB_putann (3.3.4)

3.1.2 WFDB_isigopen

```
S = WFDB_isigopen(RECORD)
```

`S`: Signal information structure array with as many elements as signals.
`RECORD`: Record name. String.

This function opens input signal files for a selected record. If RECORD begins with '+', previously opened input signal files are left open, and the record name is taken to be the remainder of RECORD after discarding the '+'. Otherwise, WFDB_isigopen closes any previously opened input signal files, and takes all of RECORD as the record name. S is a structure array with WFDB_Siginfo elements (see section 3.7.3 for an explanation of the fields), one for each signal that was opened.

Calling WFDB_isigopen also sets internal WFDB library variables that record the base time and date, the length of the record, and the sampling and counter frequencies, so that time conversion functions such as WFDB_strtim (see section 3.5.8) that depend on these quantities will work properly.

WFDB_isigopen will fill the structure array S with information about the signals in the order in which signals are specified in the 'hea' file for the record (signal numbers begin with 0). For example, the gain attributes of each signal in record '100s' can be read like this:

```
>> S = WFDB_isigopen('100s');
>> for ii = 1:length(S)
sprintf('Signal %d, gain: %d', ii-1, S(ii).gain)
```

```
end
ans =
Signal 0, gain: 200
ans =
Signal 1, gain: 200
```

An error message is produced if WFDB_isigopen is unable to open any of the signals listed in the header file, or if it cannot read the header file. It is not considered an error if only some of the signals can be opened, however.

See also: WFDB_Siginfo (3.7.3)

3.1.3 WFDB_osigopen

```
S = WFDB_osigopen(RECORD, NSIG);
```

`S`: Structure array to be filled with signal information.
`RECORD`: String. Record name from which header file will be read.
`NSIG`: Number of signals to open.

This function opens output signal files. Use it only if signals are to be written using WFDB_putvec. The signal specifications, including the file names, are read from the header file for a specified record and returned in the structure array S. Unmodified MIT or AHA database 'hea' files cannot be used, since WFDB_osigopen would attempt to overwrite the (write-protected) signal files named within. If RECORD begins with '+', previously opened output signal files are left open, and the record name is taken to be the remainder of RECORD after discarding the '+'. Otherwise, osigopen closes any previously opened output signal files, and takes all of RECORD as the record name. S is a WFDB_Siginfo structure array which, on return, will be filled with the signal specifications.

No more than NSIG (additional) output signals will be opened by WFDB_osigopen, even if the header file contains specifications for more than NSIG signals.

See also: WFDB_Siginfo (3.7.3), WFDB_putvec (3.3.2)

3.1.4 WFDB_osigfopen

```
WFDB_osigfopen(S);
```

`S`: Structure array with signal information. Create using WFDB_Siginfo (see section 3.7.3).

This function opens output signals, as does WFDB_osigopen, but the signal specifications, including the signal file names, are supplied by the caller to WFDB_osigfopen, rather than read from a header file as in WFDB_osigopen. Any previously open output signals are closed by WFDB_osigfopen. S is a

WFDB_Siginfo structure array (see section 3.7.3), with one element for each signal to be opened.

Before invoking WFDB_osigfopen, the caller must fill in the fields of the WFDB_Siginfo structure S. To make a multiplexed signal file, specify the same fname and group for each signal to be included. For ordinary (non-multiplexed) signal files, specify a unique fname and group for each signal. See section 2.4: Creating a signal file, for an illustration of the use of WFDB_osigfopen.

See also: WFDB_Siginfo (3.7.3)

3.2 Special input modes

setifreq Setting the input sampling frequency.

getifreq Determining the input sampling frequency.

setgvmode Setting the resolution for a multifrequency record.

getspf Determining the number of samples per frame.

3.2.1 WFDB_setifreq

```
WFDB_setifreq(IFREQ);
```

`IFREQ`: Input sampling frequency.

This function sets the current input sampling frequency (in samples per second per signal). It should be invoked after opening the input signals (using WFDB_isigopen), and before using any of WFDB_getvec, WFDB_getann, WFDB_putann, WFDB_isigsettime, WFDB_isgsettime, WFDB_timstr, WFDB_mstimstr, or WFDB_strtim. Note that the operation of WFDB_getframe is unaffected by WFDB_setifreq.

Use WFDB_setifreq when your application requires input samples at a specific frequency. After invoking WFDB_setifreq, WFDB_getvec resamples the digitized signals from the input signals at the desired frequency (see section 3.3.1), and all of the WFDB_tools functions that accept or return times in sample intervals automatically convert between the actual sampling intervals and those corresponding to the desired frequency.

See also: WFDB_getvec (3.3.1)

3.2.2 WFDB_getifreq

```
FREQ = WFDB_getifreq;
```

`FREQ`: Counter frequency.

This function returns the current input sampling frequency (in samples per second per signal), which is either the raw sampling frequency for the record (as would be returned by WFDB_sampfreq (see section 3.6.5), or the frequency chosen using a previous invocation of WFDB_setifreq.

See also: WFDB_sampfreq (3.6.5), WFDB_setifreq (3.2.1)

3.2.3 WFDB_setgvmode

```
WFDB_setgvmode(MODE);
```

`MODE`: String. Either 'WFDB_LOWRES' (default) or 'WFDB_HIGHRES'.

Set the mode used by WFDB_getvec when reading a multi-frequency record. If MODE is 'WFDB_LOWRES', WFDB_getvec decimates oversampled signals. If MODE is 'WFDB_HIGHRES', WFDB_getvec interpolates signals sampled at a lower frequency (repeating the last sample value).

Example: Signal 0 is sampled using 100 Hz and signal 1 using 200Hz. With WFDB_LOWRES, WFDB_getvec returns samples using 100 Hz and signal 1 is decimated from 200 Hz to 100 Hz. With WFDB_HIGHRES, WFDB_getvec returns samples using 200 Hz and signal 0 is interpolated from 100 Hz to 200 Hz.

WFDB_setgvmode also affects how annotations are read and written. If WFDB_setgvmode('WFDB_HIGHRES') is invoked before using WFDB_annopen, WFDB_getvec, WFDB_sampfreq, WFDB_strtim, or WFDB_timstr, then all time data (including the time attributes of annotations read by WFDB_getann or written by WFDB_putann) visible to the application are in units of the high-resolution sampling intervals. (Otherwise, time data are in units of frame intervals.)

3.2.4 WFDB_getspf

```
SPF = WFDB_getspf;
```

`SPF`: Samples per frame.

Unless the application is operating in WFDB_HIGHRES mode (see section 3.2.3) and has then opened a multi-frequency record, this function returns 1. For the case of a multi-frequency record being read in high resolution mode, however, WFDB_getspf returns the number of samples per signal per frame (hence WFDB_sampfreq/WFDB_getspf is the number of frames per second).

3.3 Reading and writing signals and annotations

getvec Reading input signals.

getframe Reading input signals from multifrequency records.

putvec Writing output signals.

getann Reading annotations.

putann Writing annotations.

3.3.1 WFDB_getvec

```
DATA = WFDB_getvec(NSIG);
DATA = WFDB_getvec(NSIG, NSAMP);
```

```
DATA = WFDB_getvec(NSIG, NSAMP, TSTART);
```

NSIG: Number of signals.
NSAMP: (Optional) Number of samples to read. TSTART: (Optional) Sample number of the first sample to read

This function reads samples from open input signals. Typically, we prepare to use this function by

```
S = WFDB_isigopen(record);
NSIG = length(S);
```

to open the signals for a record of choice, and to determine NSIG, the number of signals available in the record.

To read NSAMP samples of each signal, beginning at sample number TSTART:

```
DATA = WFDB_getvec(NSIG, NSAMP, TSTART);
```

The first sample of each signal has sample number 0 (not 1!).

To read the next NSAMP samples of each signal:

```
DATA = WFDB_getvec(NSIG, NSAMP);
```

This form returns up to NSAMP samples, from sample number T to sample number T+NSAMP-1, where T is the input pointer (initially 0). The input pointer is incremented by the number of samples that have been read, so that a subsequent use of WFDB_getvec returns the next NSAMP samples, etc. Use WFDB_isigsettime (see section 3.4.1) to set the input pointer directly.

If the record is not too long, read it all at once by:

```
DATA = WFDB_getvec(NSIG);
```

Note that recordings can be arbitrarily long and are often much larger than available memory; also note that there may be a very long delay if an entire record is read from a remote web server over a slow link.

3.3.2 WFDB_putvec

3.3.3 WFDB_getann

3.3.4 WFDB_putann

3.4 Non-sequential access to WFDB files

isigsettime Setting time of next samples read.

isgsettime As above, but for one signal group only.

iannsettime Setting time of next annotations read.

3.4.1 WFDB_isigsettime

3.4.2 WFDB_isgsettime

3.4.3 WFDB_iannsettime

3.5 Conversion functions

annstr, anndesc, ecgstr annotation code to string

strann, strecg string to annotation code

timstr, mstimstr time in sample intervals to HH:MM:SS or HH:MM:SS.SSS string

datstr Julian date to DD/MM/YYYY string

strdat DD/MM/YYYY string to Julian date

aduphys ADC units to physical units

physadu physical units to ADC units

adumuv ADC units to milliVolts

muvadu milliVolts to ADC units

3.5.1 WFDB_annstr

3.5.2 WFDB_anndesc

3.5.3 WFDB_ecgstr

3.5.4 WFDB_strann

3.5.5 WFDB_strecg

3.5.6 WFDB_timstr

3.5.7 WFDB_mstimstr

3.5.8 WFDB_strtim

3.5.9 WFDB_datstr

3.5.10 WFDB_strdat

3.5.11 WFDB_aduphys

3.5.12 WFDB_physadu

3.5.13 WFDB_adumuv

3.5.14 WFDB_muvadu

3.6 Miscellaneous functions

newheader Creating a 'hea' file for a new WFDB record.

wfdbquit Closing WFDB files.

iannclose Closing annotation files.

wfdbquiet Suppressing error messages from the WFDB library.

wfdberror Retrieving error messages from the WFDB library.

sampfreq Reading the sampling frequency of a WFDB record.

setsampfreq Setting the sampling frequency.

setbasetime Setting the base time.

getcfreq Functions for reading and setting counter conversion parameters.

getwfdb Reading the database path.

wfdbfile Obtaining the pathname of a WFDB file.

wfdbflush Flushing buffered output annotations and samples.

getinfo Reading info strings from a 'hea' file.

putinfo Writing info strings into a 'hea' file.

wfdbgetskew Reading intersignal skew.

wfdbsetskew Recording intersignal skew.

wfdbgetstart Reading the prolog size in a signal file.

wfdbsetstart Recording the prolog size in a signal file.

3.6.1 WFDB_wfdbquit

3.6.2 WFDB_wfdbquiet

3.6.3 WFDB_wfdbverbose

3.6.4 WFDB_wfdberror

3.6.5 WFDB_sampfreq

3.6.6 WFDB_setsampfreq

3.6.7 WFDB_setbasetime

3.6.8 WFDB_getcfreq

3.6.9 WFDB_setcfreq

3.6.10 WFDB_getwfdb

3.6.11 WFDB_wfdbflush

3.6.12 WFDB_getinfo

3.6.13 WFDB_putinfo

3.7 Creating structures

The WFDB C functions work with different structures of information about signals, annotators, and annotations. Some of the functions in the previous sections demand input of such structures. The following structure-creating functions are not wrappers to any C functions; rather, they are MATLAB m-files that create structure arrays containing the required fields, with working (although not correct in all cases) default values to avoid hard-to-find errors.

WFDB_Anninfo Create annotator structure array for input and/or output annotators

WFDB_Annotation Create annotation structure array for output annotation(s)

WFDB_Siginfo Create signal information structure for output signals

3.7.1 WFDB_Anninfo

3.7.2 WFDB_Annotation

3.7.3 WFDB_Siginfo

Support

If you believe you have found a bug, please send a report including:

- the name of the wrapper
- what input you used
- what result you got
- what result you expected
- what platform you used (CPU type, operating system name and version number, MATLAB version number, WFDB software package version number, WFDB_tools version number)

Bug reports, questions, comments, and suggestions should be addressed to:

Email: `Jonas dot Carlson at kard dot lu dot se`

Fax: +46 46 157857

Address: (postal)
Jonas Carlson
Department of Cardiology
University Hospital
SE - 221 85 LUND
Sweden