WFDB Applications Guide

Tenth Edition
(revised and with corrections for WFDB 10.6.1)
28 November 2018

George B. Moody
Harvard-MIT Division of Health Sciences and Technology

The most recent versions of the software described in this guide may be freely downloaded from PhysioNet (`http://www.physionet.org/`). For further information, write to:

George B. Moody
Massachusetts Institute of Technology
77 Massachusetts Avenue, Room E25-505A
Cambridge, MA 02139
USA

An HTML version of this guide is available at `http://www.physionet.org/physiotools/wag/`.

Contents

Introduction

Most of this guide consists of UNIX **man** pages that describe the applications included in the WFDB (Waveform Database) Software Package (and related software from PhysioToolkit). This introduction contains important information about how to interpret the material in the main sections of the guide, and about common conventions for using all of the WFDB applications that are not described in the main sections. The FAQ that follows this introduction contains additional information that will be particularly helpful if you are using MS-Windows (but it may be of interest even if you are not).

Using this Guide

The organization follows the traditional arrangement of the UNIX Reference Manual: section 1 contains programs, section 3 contains libraries, and section 5 contains file formats. In the UNIX Reference Manual, sections 2 and 4 are reserved for system calls and device interfaces respectively; these sections do not exist in this guide. Following convention, a citation such as **rdann**(1) refers to the page titled **rdann** in section 1 of this guide.

A **man** "page" may span more than one physical page, although most do not. Each **man** page in section 1 of this guide documents one or more applications, as indicated in the **NAME** section at the top. The **SYNOPSIS** appears next; it illustrates the form of the command line needed to run the application. In the synopsis, **boldface** indicates text to be typed as is, and *italics* indicate replaceable arguments; brackets ([], which are *not* to be typed) surround arguments that may be omitted, and ellipses (...) follow arguments that can be repeated. The **DESCRIPTION** sections are intentionally terse; this is a reference manual and not a tutorial introduction to the software described within. In those cases for which relevant tutorial material exists elsewhere, references appear in the **SEE ALSO** sections of each **man** page. A unique feature of this guide is the **SOURCE** section at the end of each page, which provides a URL where you may find the current version of the source(s) for each application.

On each page, the footer indicates the date when that page was last revised, and (in most cases) the version of the WFDB Software Package that was current at that time. An old date and version number do not mean that the page is out-of-date; rather they mean that the material described on that page remains current.

Under GNU/Linux, Mac OS X, or Unix, if the WFDB Software Package has been installed on your system, you can also access the information contained in the main sections of this guide using **man** and related programs. For example, to see the manual page for **rdsamp**, run the command

man rdsamp

(This also works under MS-Windows if you have installed the Cygwin package, which includes the **man** utility for formatting and reading manual pages.) In some cases you may need to add **/usr/local/man** to your **MANPATH** environment variable, in order to make these pages accessible to **man**.

An HTML version of this guide is also available (at **http://www.physionet.org/physiotools/wag/**).

Using WFDB Applications

If you have not used any of these programs before, you may need to set up your environment properly so that WFDB applications can find their input files. See **setwfdb**(1) in this guide for information about doing this; a more detailed discussion may be found in the first chapter of the *WFDB Programmer's Guide*, in the section about the database path. Most users will not need to do this, however.

Certain types of command-line arguments are used by many of the applications described in this guide. These include:

record

Where this appears, substitute the name of a WFDB record. **A record name is *not* a file name!** The first part of the name of a .hea file is the name of the record to which the .hea file belongs; so the record name corresponding to '100.hea' is '100'. For example, MIT-BIH Arrhythmia Database record names are 3-digit numbers, AHA Database record names are 4-digit numbers, and European ST-T Database record names begin with lowercase 'e', followed by a 4-digit number. Record names may contain letters, digits, and underscores. Case is significant in record names that contain letters, even in environments such as MS-Windows for which case translation is normally performed by the operating system on file names; thus 'e0104' is the name of a record found in the European ST-T Database, whereas 'E0104' is not. Once again: a record name is **not** a file name; record names never include an extension (.hea, .dat, etc.).

Wherever a record name can be supplied to a WFDB application, you may include path information if necessary. For example, if the WFDB path includes the current directory, and if the current directory includes a subdirectory named 'my_records', and that directory contains a record named 'record_23', you can supply 'my_records/record_23' as a *record* argument. See the *WFDB Programmer's Guide* for further details on record names.

Each PhysioBank database directory includes a text file named **RECORDS**, which lists the record names for all records in that directory.

annotator

Where this appears, substitute an annotator name. **Annotator names are *not* file names!** The suffix (extension) of the name of an annotation file is the annotator name for that file; so, for example, the annotator name for 'e0104.atr' is 'atr'. The special annotator name 'atr' is used to name the set of *reference annotations* supplied by the database developers. Other annotation sets have annotator names that may contain letters, digits, and underscores, as for record names.

Each PhysioBank database directory includes a text file named **ANNOTATORS**, which lists the annotator names for all annotation files in that directory.

time

Where this appears, substitute a string in *standard time format*. *Time* arguments generally specify elapsed times from the beginning of the record (for exceptions to this rule, see the section on the **strtim** function in the *WFDB Programmer's Guide*). Examples of standard time format:

2:14.875	2 minutes + 14.875 seconds
143	143 seconds (2 minutes + 23 seconds)
4:02:01	4 hours + 2 minutes + 1 second
4:2:1	same as above
s12345	12345 sample intervals
e	time of the end of the record

signal

Where this appears, substitute a signal number or (in most cases) a signal name. Signal numbers are integers; the first signal in each record is signal 0. In printed documentation for the databases, signals always appear with signal 0 at the top, signal 1 beneath, etc. Signal names are the strings printed by **signame**(1).

signal-list

Where this (or '*signal* ...') appears, you may specify more than one signal in any desired order; separate the signal numbers or names using spaces. Unless otherwise noted, a signal may appear more than once, or not at all, in a signal list. In most cases, the end of the signal list is unambiguous (since signal numbers are never negative, and signal names rarely if ever begin with '-', an option argument beginning with '-' is a reliable indicator). In unusual cases, you may need to arrange options so that the signal list is at the end of the command, or so that it is followed by an argument that cannot be interpreted as a signal number.

Frequently Asked Questions (and Frequently Exclaimed Exclamations)

I double-clicked on the program icon, and nothing happens! I typed the program name in the 'Run...' dialog, and nothing happens!

Don't do this!

With few exceptions, PhysioToolkit applications run in **text mode** (i.e., they do not include a graphical user interface). These programs are intended to be run within a terminal emulator using a command-line interface. In most cases, if you attempt to run them by clicking on their icons or names, or by entering the program name in the MS-Windows **Run...** dialog box, these programs will open a DOS box, print a usage summary, and exit, usually much too fast for you to read anything.

By far the best way to use these programs under MS-Windows is to install a Unix-compatible terminal emulator and shell in which to run them. The best of these is also free; if you have not already done so, download and install the Cygwin software package from **http://www.cygwin.com/**. This package includes **bash**, the GNU Bourne Again Shell and a terminal emulator in which to run it. After a standard installation of Cygwin, you can launch a terminal emulator and **bash** by clicking on the Cygwin icon that will have been installed on your desktop.

If you do not wish to use Cygwin, it is possible to run these applications within a DOS box, but there are many limitations of **command.com** that may prove frustrating. In particular, **command.com** supports a relatively small space for environment variables that is not secure against buffer overruns, and has idiosyncratic filename globbing behavior.

What does the message "init: can't open header for ..." mean?

This message can be produced by any application linked to the WFDB library, including **rdsamp**(1) and **rdann**(1). In order to read data files, these applications need to find a header (**.hea**) file for the input record you specify. The message indicates that the header file was not found in any of the expected places, or that it was unreadable. There are three common reasons why this can happen:

- The *record* name supplied to the application is not correct. Record names are **not** file names (if this doesn't sound familiar yet, go back and read the introduction again). If you wish to read, for example, a signal file named **slp60.dat** using **rdsamp**, you must specify the name of the record to which this file belongs (**slp60**) after the **-r** option, and not the name of the file itself. Whatever follows "init: can't open header for ..." is what the application thinks is the name of the record you wish to read. Also, be aware that case matters in record names, even under operating systems that ignore case in file names. Thus "**SLP60**" is not a valid record name; "**slp60**" is.
- The header file is missing. If you download signal (**.dat**) or annotation (**.atr**, **.qrs**, etc.) files, be sure to download the corresponding **.hea** files from the same locations.
- The list of locations to be searched does not include the location of the header file. WFDB applications find their input files by searching a list of locations specified by the WFDB path (the environment variable **WFDB**, or a default list of locations if WFDB has not been set). The WFDB path normally includes the current directory, but this may not be true if the WFDB path has been modified; the current directory must appear explicitly (either as a "." or as an empty component in the path) in order to be included in the list of locations to be searched. For further information, see "The Database Path and Other Environment Variables" in the *WFDB Programmer's Guide*.

How can I save the output of ... in a file?
How can one program read another's output?

If you are running programs from a command prompt (by typing commands into a terminal emulator window or an MS-DOS box), these things can be done easily.

If you have ever used GNU/Linux, Unix, or MS-DOS, you may have captured the output of a program by *redirecting* it to a file, like this:

foo >bar

The > operator redirects **foo**'s standard output (which would normally appear on-screen) into a file named **bar**. If **bar** exists already, its contents are replaced. If you wish to append **foo**'s output to whatever is already contained in **bar**, use a command such as this instead:

foo >>bar

There is an analogous operator that arranges for a program's standard input (which would normally be read from whatever you type on the keyboard) to be read from a file instead:

baz <bar

Here, the < operator arranges for **baz** to read its input from a file named **bar**. If **bar** was created by **foo**, then this command allows **baz** to read **foo**'s output.

You can combine input and output redirection in a single command using the pipe (|) operator:

foo | baz

This command runs **foo** and sends its standard output directly to **baz**, without requiring an intermediate file. True multitasking operating systems such as Unix and GNU/Linux allow both programs to run (apparently) simultaneously; under MS-DOS or MS-Windows, the first program runs to completion before the second one begins execution.

You can use these techniques whenever you run programs from a command prompt, whether those programs are among those available here or obtained from some other source. You can use the same techniques with programs you write yourself; the only requirement is that your programs must read from the standard input and write to the standard output (i.e., they must not attempt to bypass the standard input/output mechanism by reading directly from the keyboard or writing directly to the screen).

These operators (>, >>, <, and |) are supported by all shells (command interpreters) under Unix, GNU/Linux, and MS-DOS (including those that run within MS-DOS boxes or other types of terminal emulators under MS-Windows). For further information, please refer to the documentation for your shell or command interpreter.

Where else can I find answers to my questions about this software?

If you haven't read the introduction to this guide yet, do so now. It answers many frequently asked questions by describing the common behavior of many of the WFDB applications. It also describes the typographic and organizational conventions used in the remainder of this guide.

Many more questions are asked and answered in the PhysioNet FAQ (http://www.physionet.org/faq.shtml).

NAME

a2m, ad2m, ahaconvert, ahaecg2mit, m2a, md2a – converting between AHA DB and WFDB formats

SYNOPSIS

To read from an AHA DB DVD:
ahaecg2mit [**-s**] *ahafile***.*** ...
To read from an AHA DB CD:
ahaconvert *ahafile***.cmp** ...
To read from an AHA DB floppy disk or 9-track tape:
a2m -i *ahafile* **-r** *record* **-a** *annotator* [*options* ...]
ad2m -i *ahafile* **-r** *record* [*options* ...]
To convert a WFDB record to AHA tape format:
m2a -r *record* **-a** *WFDB-annotator AHA-annotator* [*options* ...]
md2a -o *ahafile* **-r** *record* [*options* ...]

DESCRIPTION

The AHA Database for Evaluation of Ventricular Arrhythmia Detectors (AHA DB) has been distributed since 1983 by ECRI (http://www.ecri.org), in at least four formats:

single-file (.txt) format
Developed by ECRI for distributions of the AHA DB on DVDs (ca. 2012) as a successor to the earlier .ecg format (below); this format can be read by **ahaecg2mit**.

single-file (.ecg) format
Developed by ECRI for distributions of the AHA DB on DVDs (ca. 2008); this format can be read by **ahaecg2mit**.

compressed (.cmp and .ano) format
Previously developed by ECRI for distributions of the AHA DB on floppy disks (ca. 1990) and CDs (ca. 1995); this format can be read by **ahaconvert** (using **a2m** and **ad2m**).

tape format
Originally specified by the creators of the AHA DB at the Biomedical Computing Laboratory (BCL) at Washington University in St. Louis. This format was also used for tape distributions of the MIT-BIH Arrhythmia Database from 1980-1989; it can be read by **a2m** and **ad2m**, and written by **m2a** and **md2a**.

The AHA DB consists of a **development** set of 80 records (which were created by the BCL between 1976 and 1983 and have been distributed by ECRI since then) and a **test** set of 75 records (also created by the BCL between 1976 and 1983, but not distributed until about 20 years later). Each record contains two simultaneous ECG signals that have been digitized for three hours continuously, and beat annotations for the final 30 minutes of the signals in each record. The records have been distributed in two versions: a **long version** (records named n0nn and n1nn) containing the full three hours of signals, and a **short version** (records named n2nn and n3nn) containing only the final 35 minutes of signals (including all of the annotated beats).

ECRI currently supplies the AHA DB only on DVDs, so the tape and compressed formats are primarily of historical interest. The programs described below convert these formats into WFDB (also known as PhysioBank or MIT) format. Long version input files can be converted in their entirety, or these programs can create short version records from either long or short version inputs. The last two programs below convert WFDB records to AHA tape format (conversion to AHA DB DVD and CD/floppy disk distribution formats is not supported). All of these programs print a brief usage summary if invoked with no command-line arguments, or with a **-h** option.

Note that records in WFDB format can be excerpted and reformatted in more generally useful ways using **snip**(1) or **xform**(1).

DVD FORMAT

ahaecg2mit

Use **ahaecg2mit** to convert .txt or .ecg files from an AHA DB DVD into WFDB-compatible records. One or more input filenames may be supplied as command-line arguments; each specified input file is converted into a WFDB record, including a .hea (header) file, a .dat (signal) file, and a .atr (reference annotation) file. If the first command-line argument is **-s**, then **ahaecg2mit** produces short-form records with the correct (n2nn or n3nn) record names.

If the .txt or .ecg files are not in the current directory, give their full pathnames. The output files are always written to the current directory, so be sure that the current directory is writeable (it should not be the DVD) and that has sufficient free space (roughly 8 Mb per long version record, or 1.6 MB per short version record).

OLDER FORMATS

ahaconvert

Use **ahaconvert** to convert one or more records from an AHA DB CD-ROM into WFDB format. Run **ahaconvert** without any command-line arguments for instructions, or see the examples below. Note: **ahaconvert** is a shell script; to use it successfully, you will need to have a shell (standard with all versions of Unix, and included in the free Cygwin package for MS-Windows) as well as **ad2m** and **a2m**, which perform the actual work of the conversion.

a2m

Use **a2m** to convert AHA-format annotation files from tapes, floppy disks, or CDs into WFDB format. Options for **a2m** include:

-s *time* Shift annotations forward by the specified *time* (default: no shift for type 0 input files, 5 minutes for type 1, 2 hours and 30 minutes for type 2; for type 3, the default is 5 minutes if *record* is of the form *n*2*nn* or *n*3*nn*, or 2 hours and 30 minutes if *record* is of the form *n*0*nn* or *n*1*nn*).

-t *type* Convert an input file of the specified *type* (0: a file produced by a WFDB application using **putann** and **WFDB_AHA_WRITE** mode; 1: an AHA DB 'short format' tape file; 2: an AHA DB 'long format' tape file; 3: an AHA DB compressed (**.ano**) CD-ROM or floppy disk file). Input files of types 1, 2, and 3 are assumed to contain annotation times in milliseconds, which are converted to sampling intervals based on an assumed sampling frequency of 250 Hz. Default: type 3 is assumed if *ahafile* ends with **.ANO** or **.ano**; type 0 is assumed otherwise.

ad2m

Use **ad2m** to convert AHA-format signal files from tapes, floppy disks, or CDs into WFDB format. Options for **ad2m** include:

-c Convert an AHA DB compressed (**.cmp**) floppy disk file (this is the default if *ahafile* ends with **.CMP** or **.cmp**, otherwise **ad2m** assumes that the input is a file in AHA DB tape format).

-f *time* Begin converting at the specified *time* relative to the beginning of the input file (default: 0, i.e., at the beginning of the input file)

-t *time* Stop converting at the specified *time* relative to the beginning of the input file (default: 35 minutes after the starting time if *record* is of the form *n*2*nn* or *n*3*nn*, 3 hours if *record* is of the form *n*0*nn* or *n*1*nn*, or the end of the input file, whichever comes first).

m2a

Use **m2a** to convert WFDB-format annotation files into AHA tape format. Options for **m2a** include:

-s *time* Shift annotation times backward by the specified *time*, and convert them from sample intervals to milliseconds.

md2a

Use **md2a** to convert WFDB-format signal files into AHA tape format. Options for **md2a** include:

-n *new-record*
: Create a new header file for the AHA-format output signal file, so that it may be read as record *new-record*.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

EXAMPLES

AHA Database DVD

If the DVD is accessible as **/media/dvd/**, either

ahaecg2mit -s /media/dvd/*.txt

or (for DVDs written in the older format):

ahaecg2mit -s /media/dvd/*.ecg

makes a complete set of short-version records in the current directory. (Omit the **-s** to make a set of long-version records instead.) Under Windows, within a Cygwin window, the DVD is accessible as **/cygdrive/d/** (or **/cygdrive/e/**, etc., depending on the drive letter that Windows has assigned), so the same task can be done under Windows by

ahaecg2mit -s /cygdrive/d/*.txt

AHA Database CD

AHA DB CDs contain both long and short versions of each record. In most cases, you will want to convert only one version of each record. To convert the short-version records only, if the contents of the CD-ROM are available at **/mnt/cdrom**, type:

ahaconvert /mnt/cdrom/?[23]??.cmp

(The pattern '**?[23]??**' matches the record names of the short-version records.)

To convert the long-version records only, type:

ahaconvert /mnt/cdrom/?[01]??.cmp

AHA DB floppy disk

To make a version of AHA DB record 1201 in WFDB format, given the distribution floppy disk, copy the files **1201.ano** and **1201.cmp** to the current directory, then type:

ad2m -i 1201.cmp -r 1201 -c
a2m -i 1201.ano -r 1201 -a atr -t 3

These commands produce files **1201.dat** (the signal file), **1201.hea** (the header file), and **1201.atr** (the reference annotation file), all in the current directory. Run **ad2m** first, so that the new header file is available for the use of **a2m**. (In this example, note that the options '-r 1201', '-c', and '-t 3' are redundant unless you have renamed the input files, since **ad2m** and **a2m** recognize the record name and file types from the suffixes otherwise.)

AHA DB short version tape

To obtain the same files given a 'short version' 9-track distribution tape, copy the second and third files from the tape into files **1201.tap** and **1201.ann** in the current directory, then type:

ad2m -i 1201.tap -r 1201
a2m -i 1201.ann -r 1201 -a atr -t 1

The names for the files copied from the tape are arbitrary, but do not use names of files to be generated by **ad2m** or **a2m** (see the previous example). Note that the first and fourth files on the distribution tape contain an 'id' block, which can be read by **readid** (a program included in the **convert** directory of the WFDB Software Package) to verify the record name. Distribution tapes that contain more than one record contain additional sets of four files, always in the same order within each set.

AHA DB long version tape

To make a version of the three-hour AHA DB record 1001 in WFDB format, given the 'long version' distribution tape, copy the second and third files from the tape into files **1001.tap** and **1001.ann** in the current directory, then type:

ad2m -i 1001.tap -r 1001 -t 3:0:0
a2m -i 1001.ann -r 1001 -a atr -t 2

The **-t 3:0:0** option is necessary to prevent **ad2m** from truncating the signal file after the first 35 minutes.

Converting AHA DB long version tapes to short version records

To make a version of AHA DB record 1201 in WFDB format, given a 'long version' 9-track distribution tape containing the corresponding three-hour record 1001, copy the second and third files from the tape into files **1001.tap** and **1001.ann** in the current directory, then type:

ad2m -i 1001.tap -r 1201 -f 2:25:0
a2m -i 1001.ann -r 1201 -a atr -t 1

In this case, the **-f** option instructs **ad2m** to skip the first two hours and 25 minutes of the 'long-version' AHA signal file, and to reformat the remainder (equivalent to the 35-minute 'short-version' record). The **-t 1** option is used with **a2m** even though its input file comes from a 'long-version' tape, because the annotation times must be shifted only by the amount necessary for a 'short-version' tape in this case.

Sharing signal files for long version and short version AHA DB records

To keep both versions (1001 and 1201) on-line, make the long version first (see above), then type:

a2m -i 1001.ann -r 1201 -a atr -t 1

to make a short version reference annotation file. Continue (under UNIX) by:

cp 1001.hea 1201.hea

or (under MS-DOS) by:

copy 1001.hea 1201.hea

and edit **1201.hea**, replacing '1001' in the first line (only!) with '1201', and replacing '212' in the second and third lines by '212+6525000' (see the description of the 'byte offset' field in **header**(5)). Although each version needs its own header and reference annotation files, the long-version signal file can be shared with the short version, allowing a substantial savings in storage requirements. Note that WFDB application programs that read the 'short version' record 1201 signal file may report signal checksum errors at the end of the record, unless you also recalculate the signal checksums (easily done using **snip**(1) to copy the record; delete the copy once the checksums have been obtained).

AVAILABILITY

These programs are provided in the **convert** directory of the WFDB Software Package. Run **make** in that directory to compile and install them if they have not been installed already.

SEE ALSO

snip(1), **xform**(1), **wfdb**(3), **header**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/convert/a2m.c
http://www.physionet.org/physiotools/wfdb/convert/ad2m.c
http://www.physionet.org/physiotools/wfdb/convert/ahaconvert
http://www.physionet.org/physiotools/wfdb/convert/ahaecg2mit.c
http://www.physionet.org/physiotools/wfdb/convert/m2a.c
http://www.physionet.org/physiotools/wfdb/convert/md2a.c

NAME

ann2rr, rr2ann – convert annotation files to interval lists and vice versa

SYNOPSIS

ann2rr -r *record* **-a** *annotator* [*options ...*]
rr2ann -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

These programs are typically used to obtain RR interval series from ECG annotation files, or to create an annotation file from such a series, but they have a wider range of uses.

ann2rr

Use **ann2rr** to extract a list of intervals, in text format, from an annotation file. By default, the intervals are listed in units of sample intervals (use **sampfreq**(1) to determine the sampling frequency of the input record if necessary). Options for **ann2rr** include:

-A Print all intervals between annotations. By default, **ann2rr** prints only RR intervals (those between QRS (beat) annotations). This option overrides the **-c** and **-p** options.

-c Print intervals between consecutive valid annotations only. (See discussion below.)

-f *time* Begin at the specified *time*. By default, **ann2rr** starts at the beginning of the record.

-h Print a usage summary.

-i *format*
Print intervals in the specified *format*. By default, intervals are printed in units of sample intervals. Other *format*s include **s** (seconds), **m** (minutes), **h** (hours), and **t** (time interval in hh:mm:ss format). Formats **s**, **m**, and **h** may be followed by an integer between 0 and 15 inclusive, specifying the number of decimal places (default: 3). For example, use the option **-is8** to obtain intervals in seconds with 8 decimal places.

-p *type* [*type ...*]
Print intervals ended by annotations of the specified *types* only. The *type* arguments should be annotation mnemonics (e.g., **N**), as normally printed by **rdann**(1) in the third column. More than one **-p** option may be used in a single command, and each **-p** option may have more than one *type* argument following it. If *type* begins with "-", however, it must immediately follow **-p** (standard annotation mnemonics do not begin with "-", but modification labels in an annotation file may define such mnemonics).

-P *type* [*type ...*]
Print intervals begun by annotations of the specified *types* only.

-t *time* Stop at the specified *time*.

-v *format*
Print final times (the times of occurrence of the annotations that end each interval). This option accepts all of the *format*s defined for **-i**, as well as **T** (to print the date and time in [hh:mm:ss dd/mm/yyyy] if the starting time and date have been recorded in the header file for *record*). If this option is chosen, the times appear at the end of each line of output.

-V *format*
Print initial times (the times of occurrence of the annotations that begin each interval). Any of the *format*s usable for the **-v** option may be used with **-V**. If this option is chosen, the times appear at the beginning of each line of output.

-w Print final annotations (the types (**N**, **V**, etc., as for **-p** above) of the annotations that end each interval), immediately following the intervals in each line of output.

-W Print initial annotations (the types of the annotations that begin each interval), immediately before the interval in each line of output.

The **-c** option, used without the **-p** option, causes **ann2rr** to filter out intervals between beats that have

intervening non-beat annotations, such as rhythm or signal quality change annotations. Used with the **-P** and **-p** options, the **-c** option causes **ann2rr** to reject intervals between annotations of the type(s) specified by **-p** if there are annotations of any other types intervening; thus, for example, "**-c -P N -p N**" yields only intervals between consecutive normal beats, and intervals between pairs of normal beats surrounding an ectopic beat are discarded from the output. As another example, "**-c -P N -p V**" yields premature ventricular coupling intervals only (a coupling interval is the interval between a normal beat and an immediately following premature ventricular contraction).

The default output contains a single column of intervals only; by using the **-v**, **-V**, **-w**, and **-W** options, up to five columns, separated by tabs, may be output. The order of the columns is fixed (initial times, initial annotations, intervals, final annotations, final times).

rr2ann

Use **rr2ann** to create an annotation file from the standard input, which should usually be a list of intervals in the format produced by **ann2rr**. (For exceptions, see **-T**, **-w**, and **-x** below.) The first token on each line is taken as an interval, and (if the **-w** option is present) the second token is taken as an annotation mnemonic; anything else on the same line is ignored, as are empty lines, spaces and tabs at the beginning of a line, non-numeric tokens and anything following them on the same line, negative intervals, and zero intervals. The output consists of a binary annotation file (*record.annotator*), and (if it does not exist already) a text header file (*record*.hea). Options for **rr2ann** include:

-F *frequency*
: Assume the specified sampling *frequency*. This option has no effect unless it is necessary for **rr2ann** to create a header file; in this case, a sampling frequency of 250 Hz is assumed if the **-F** option is omitted.

-h Print a usage summary.

-T Interpret the input as times of occurrence, rather than as intervals.

-w Set each annotation type from the mnemonic (**N**, **V**, etc.) in the second column of the input (in the format produced by **ann2rr** using its **-w** option).

-x *n* Multiply input by *n* to obtain intervals (or, if **-T** is also used, times of occurrence) in units of sample intervals). Default: *n* = 1.

Note that **wrann**(1) also provides a way to generate an annotation file from text. Unlike that of **rr2ann**, **wrann**'s input format permits specifying annotation subtypes and other fields.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

record.hea	header file
record.annotator	annotation file

AVAILABILITY

These programs are provided in the *app* directory of the WFDB Software Package. Run **make** in that directory to compile and install them if they have not been installed already.

The PhysioNet ATM (http://physionet.org/cgi-bin/ATM) provides web access to **ann2rr** (select **Show RR intervals as text** from the Toolbox).

SEE ALSO

rdann(1), **sampfreq**(1), **setwfdb**(1), **wrann**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/ann2rr.c
http://www.physionet.org/physiotools/wfdb/app/rr2ann.c

NAME

bxb – ANSI/AAMI-standard beat-by-beat annotation comparator

SYNOPSIS

bxb -r *record* **-a** *reference-annotator test-annotator* [*options ...*]

DESCRIPTION

Using options **-C**, **-L**, or **-S**, **bxb** implements the beat-by-beat comparison algorithms described in ANSI/AAMI EC38:1998, the *American National Standard for Ambulatory ECGs*, and in ANSI/AAMI EC57:1998, the *American National Standard for Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms*. **bxb** is the reference implementation of these algorithms, and must be used to obtain the beat-by-beat performance statistics cited in EC38 and EC57 in order to be in compliance with these standards (see EC38, section 5.2.14, and EC57, section 4.2). The **-C**, **-L**, and **-S** options also gather statistics on RR interval errors, which were considered for inclusion in EC38, but were eventually dropped from it.

Input to this program consists of two annotation files associated with the same *record*. One of these is designated the *reference* annotation file, the other the *test* annotation file (called the 'algorithm' annotation file in EC38 and in EC57).

Options include:

-c *file* Append condensed reports (EC57 Table A.2.1 format) to *file*.

-C *file* As for **-c**, but report RMS RR interval error and SVEB statistics also.

-f *time* Begin the comparison at the specified *time* (default: 5 minutes after the beginning of the record).

-h Print a usage summary.

-l *file1 file2*
Append line-format reports (EC57 Tables A.2 and A.3 format) to *file1* and *file2* respectively (see below).

-L *file1 file2*
As for **-l**, but report RMS RR interval error and SVEB statistics also.

-o Generate an output annotation file (see below).

-O Generate an expanded output annotation file (see below).

-s *file* Append standard reports (EC38, section 5.2.14, and EC57, Table 3 format) to *file*.

-S *file* As for **-s**, but report RMS RR interval error and SVEB statistics also.

-t *time* Stop the comparison at the specified *time* (default: the end of the record if it is defined, the end of the reference annotation file otherwise; if *time* is 0, the comparison ends when the end of either annotation file is reached).

-v Verbose mode (list all beat label discrepancies; see below).

-w *time* Set the *match window* (default: 0.15 seconds; see below).

The statistics gathered by **bxb** are based on tallies of 'matching' annotations in the reference and test annotation files. Matching annotations need not have exactly equal annotation times; the *match window* specifies the maximum absolute difference in annotation times that is permitted for matching annotations. **bxb** measures the total shutdown time in the test annotation file as the sum of all intervals that begin with a 'shutdown' annotation and that end with a 'resume' annotation. (If a period of shutdown does not end before the end of the record, the creator of the annotation file should nevertheless write a 'resume' annotation at the end of the record, in order to permit correct shutdown accounting.) This program follows the convention for 'shutdown' and 'resume' annotations adopted for reference annotation files of the European ST-T database, a convention compatible with that established for the MIT-BIH Arrhythmia Database: 'shutdown' annotations are **NOISE** annotations with bits 4 and 5 (i.e., the '16' bit and the '32' bit) of the subtype field both set; 'resume' annotations are **NOISE** annotations with any other subtype. The convention used in AHA Database reference files, in which unreadable intervals are marked by only one 'shutdown'

annotation placed near the middle of the interval, is also acceptable; in this case, shutdown is assumed to begin 150 ms after the previous annotation, and it is assumed to end 150 ms before the following annotation.

At most one of **-c**, **-C**, **-l**, **-L**, **-O**, **-s**, and **-S** can be given as an option. If '**-**' is given as a *file* argument, reports are written on the standard output. If no options are specified, **bxb** writes standard reports on the standard output (equivalent to using the option **-s -**). The output generated by selecting **-l** or **-L** includes column headings only if a *file* other than '**-**' is specified, and only if the specified *file* does not already exist. In this way, **bxb** can be used repeatedly to build up a line-format table for multiple records, for further processing by **sumstats**(1).

The **-o** option produces an output annotation file with annotator name **bxb**. The output annotation file contains exact copies of all of the test annotator's beat labels that match those of the reference annotator, as well as **NOTE** annotations that describe all mismatches. Mismatched annotation types are mapped into the AAMI 'test label' mnemonics (**N**, **V**, **F**, **Q**, **O**, and **X**; if the **-C**, **-L**, or **-S** option is also specified, the mnemonics also include **S**). The 'aux' field of each **NOTE** annotation indicates the element of the confusion matrix in which the mismatch is tallied (e.g., **Nv** represents an eventcalled a normal beat by the reference annotator and a ventricular ectopic beat by the test annotator). **NOTE** annotations that correspond to beats missed by the test annotator are placed at the sample indicated by the reference annotation; all others are placed at that indicated by the test annotation.

The **-O** option produces a similar output annotation file, in this case containing not only beat labels but all others as well. No summary report is produced if **-O** is specified. **NOTE** annotations produced using **-O** contain unmapped annotation mnemonics from the input annotation files. This option, if used together with **-f 0 -w 0**, identifies all discrepancies between a pair of annotation files. It can be especially useful for developing reference annotation files for new records.

The **-v** option specifies that each beat label mismatch is described on the standard output in a format similar to:

N(120188)/V(120191)

where the letters indicate the AAMI mnemonics corresponding to the reference and test annotators' beat labels, and the numbers indicate the *time* fields (sample numbers) of the reference and test annotations respectively. Note that **O** and **X** mnemonics are generated by **bxb** as placeholders for missing beat labels; you will not find them in the input annotation files.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

DIAGNOSTICS

non-standard comparison selected

The **-f**, **-O**, **-t**, and **-w** options modify the comparison algorithm used by **bxb** in ways not permitted by EC38 or EC57. These options are provided for the use of developers, who may find them useful for obtaining a more detailed understanding of algorithm errors.

SEE ALSO

ecgeval(1), **epicmp**(1), **mxm**(1), **rxr**(1), **setwfdb**(1), **sumstats**(1)

Evaluating ECG Analyzers (in the *WFDB Applications Guide*)

American National Standard ANSI/AAMI EC38:1998, Ambulatory Electrocardiographs

American National Standard ANSI/AAMI EC57:1998, Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms

The last two publications are available from AAMI, 1110 N Glebe Road, Suite 220, Arlington, VA 22201 USA (http://www.aami.org/).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/bxb.c

NAME

calsig – calibrate signals of a WFDB record

SYNOPSIS

calsig -r *record* [*options ...*]

DESCRIPTION

calsig (formerly known as **calibrate**) rewrites the header file for a WFDB record, setting the gain and baseline fields based on measurements it makes, and setting the units fields based on input from the user or from a calibration file. Normally, **calsig** is used by specifying a time interval for the measurements; best results will be achieved if the specified interval is restricted to one or more square-wave calibration pulses in each signal to be calibrated, although sine-wave pulses may be usable if the sampling frequency and/or ADC resolution is high enough.

The program constructs a smoothed amplitude histogram for each signal and identifies its two principal modes. Initially, each bin of the histogram counts the number of samples in the analysis interval for which the amplitude has a specified value. The histogram is smoothed by applying a low-pass filter that replaces the contents of each bin by a weighted sum of fifteen bins centered on the bin of interest. The two principal modes in the smoothed histogram must be separated by at least one bin with a count that is less than one-eighth the count of the larger mode. If this criterion is not satisfied for a given signal, **calsig** warns the user and does not adjust the gain or baseline for the affected signal.

If a signal list is specified using the **-s** option (see below), only the specified signals are calibrated, and the gain, baseline, and units fields for any other signals are left unchanged. Thus, if calibration pulses are not simultaneously available in all signals to be calibrated, **calsig** may be run repeatedly with different time intervals and signal lists.

Options include:

-c *file* Obtain calibration pulse specifications from the specified *file* (see **wfdbcal**(5); default: obtain this information from the file specified by the environment variable **WFDBCAL**, or interactively).

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a usage summary.

-q Instead of using the standard method for calibration, get a 'quick-and-dirty' estimate by taking the signal amplitudes at the starting and ending times (as specified by **-f** and **-t**) as representative of the low- and high-amplitude phases of the calibration pulse. Use this option only if the standard method fails; the standard method is more accurate.

-Q Use an alternate 'quick-and-dirty' estimate based on the range of signal amplitudes over the interval specified by **-f** and **-t**. This method may be easier to use than **-q** for signals with significant high-frequency content, since it does not require precise location of signal extrema. As noted above, the standard method is more accurate if it can be used.

-s *signal-list*
Calibrate only the signals named in the *signal-list* (one or more input signal numbers or names, separated by spaces; default: calibrate all signals).

-t *time* Process until the specified *time* in *record* (default: 1 second after the starting time).

-v Ask for calibration pulse limits (default: read limits from the calibration file).

ENVIRONMENT

It may be necessary to set and export the shell variables **WFDB** and **WFDBCAL** (see **setwfdb**(1)).

Calibration files must be located in one of the directories named in **WFDB**, the database path. If the environment variable **WFDBCAL** is set, it names a calibration file that will be read unless the **-c** option is used to specify a different calibration file. At most one calibration file is read; if more than one **-c** option is given, only the last one is effective. If the calibration file does not contain an entry for the type of signal to be calibrated, **calsig** obtains the information from the user interactively. If the calibration file contains two or more entries for the same signal type, only the first entry is used.

SEE ALSO

setwfdb(1), **wfdbcal**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/calsig.c

NAME

coherence – estimate coherence and cross-spectrum of two time series

SYNOPSIS

coherence -i *file* [*options ...*]

DESCRIPTION

coherence estimates the coherence and cross-spectrum of a pair of real-valued time series; as a byproduct of its calculation of coherence, it also estimates the autospectra of each of its input time series. The *file* argument specifies the name of a text file containing the samples of the series in two columns. The standard output contains five columns of numbers (optionally preceded by column headings), which are frequency (Hz), coherence, cross-spectral power (dB), autospectral power (dB) of the first time series, and autospectral power (dB) of the second time series.

This program is based on a Fortran program by C.R. Arnold, G.C. Carter, and J.F. Ferrie, as described in 'A coherence and cross-spectral estimation program', by G.C. Carter and J.F. Ferrie, in *Programs for Digital Signal Processing*, edited by the Digital Signal Processing Committee of the IEEE ASSP Society (New York: IEEE Press, 1979). The functions fft842() and its auxiliary functions r2tx(), r4tx(), and r8tx(), are based on Fortran subroutines by G.D. Bergland and M.T. Dolan, as described by them in 'Fast Fourier transform algorithms', also included in *Programs for Digital Signal Processing*.

Options are:

-f *frequency*
Specify the sampling frequency in Hz (default: 250).

-n *n* Process the input in overlapping chunks of *n* samples (default: 1024). For best results, *n* should be a power of two.

-v Print column headings.

-x *sx sy* Specify multiplicative scale factors for the two time series (defaults: 1). A reasonable choice is to use the reciprocals of the standard deviations of the respective time series if these differ significantly.

Note that the scale factors generally have little or no visible effect on the coherence or on the shape of the spectra. The choice of chunk size (using the **-n** option) will have a significant effect; some experimentation may be needed to determine an appropriate chunk size in each case.

SEE ALSO

fft(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/psd/coherence.c

NAME

dfa – detrended fluctuation analysis

SYNOPSIS

dfa [*option ...*]

DESCRIPTION

The method of detrended fluctuation analysis (DFA) has proven useful in revealing the extent of long-range correlations in seemingly irregular time series.

Briefly, the time series to be analyzed is first integrated. Next, the integrated time series is divided into boxes of equal length, *n*. In each box of length *n*, a least squares line (or polynomial curve of order *k*) is fit to the data (representing the trend in that box). Next, we detrend the integrated time series by subtracting the local trend in each box. The root-mean-square fluctuation of this integrated and detrended time series is calculated and denoted as *F(n)*.

This computation is repeated over all time scales (box sizes), from *n = minbox* to *n = maxbox*, to characterize the relationship between *F(n)*, the average fluctuation, and *n*, the box size. Typically, *F(n)* will increase with box size *n*. A linear relationship on a log-log plot indicates the presence of power law (fractal) scaling. Under such conditions, the fluctuations can be characterized by a scaling exponent, i.e., the slope of the line relating *log[F(n)]* to *log[n]*.

This program performs detrended fluctuation analysis on a sequence of data read from the standard input (which should contain a single column of numbers in text format). The standard output contains two columns of numbers, which are the base 10 logarithms of *n* and *F(n)*. Note that **dfa** does *not* compute a scaling exponent; to do so, fit the output to a line and measure its slope.

Options may include:

-d *k* Detrend the data using a polynomial of degree *k* (1: linear, 2: quadratic, etc.). Default: *k* = 1 (linear detrending).

-h Print a usage summary and exit.

-i Do not integrate the input series. Use this option if the input series is already integrated (for example, if it represents times of occurrence rather than intervals).

-l *minbox*
Set the smallest box width. The default, and the minimum allowed value for *minbox*, is *2k + 2* (where *k* is determined by the **-d** option, see above).

-s Perform a sliding window DFA (measure the fluctuations using all possible boxes at each box size). By default, fluctuations are measured using non-overlapping boxes only. Using the **-s** option will make the calculation much slower.

-u *maxbox*
Set the largest box width. The default, and the maximum allowed value for *maxbox*, is one-fourth the length of the input series.

SEE ALSO

The DFA method was first proposed in Peng C-K, Buldyrev SV, Havlin S, Simons M, Stanley HE, Goldberger AL. Mosaic organization of DNA nucleotides. *Phys Rev E* 1994;**49**:1685-1689.

A detailed description of the algorithm and its application to physiologic signals can be found in Peng C-K, Havlin S, Stanley HE, Goldberger AL. Quantification of scaling exponents and crossover phenomena in nonstationary heartbeat time series. *Chaos* 1995;**5**:82-87.

AVAILABILITY

dfa is available as part of PhysioToolkit under the GPL (see **SOURCE** below).

AUTHORS

JE Mietus (joe@physionet.org), C-K Peng, and GB Moody, based on C-K Peng's original Fortran implementation.

SOURCE

http://www.physionet.org/physiotools/dfa/dfa.c

NAME

ecgeval – generate and run ECG analyzer evaluation script

SYNOPSIS

ecgeval

DESCRIPTION

This program generates a Bourne shell (**sh**(1)) script under UNIX, or a batch file under MS-DOS, to compare a set of test annotation files with a set of reference annotation files and a set of reference heart rate measurement files using the programs **bxb**(1), **rxr**(1), **mxm**(1), and **epicmp**(1), and then to produce summary reports by passing the outputs of these programs to **sumstats**(1) and **plotstm**(1).

ecgeval asks interactively for the annotator names, the name of the database to be used, and which optional analyzer outputs are to be evaluated. It then creates the evaluation script, and offers the user a choice of running the script immediately, or exiting (in order to review and perhaps edit the script before running it).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

dblist

This file, which should be located in one of the directories named by **WFDB**, contains a list of the available databases. Each entry is a line containing three tab-separated fields: the short name for the database, the name of a file (which must also be in one of the directories named by **WFDB**) containing a list of the record names for the database, and a longer name for the database. Empty lines and lines beginning with '#' are ignored. The version of this file distributed with the WFDB software package contains:

MIT DB	mitlist	MIT-BIH Arrhythmia Database
MITx DB	mitxlist	MIT-BIH Arrhythmia Database (excluding paced records)
AHA DB	ahalist	AHA Database for Evaluation of Ventricular Arrhythmia Detectors
AHAx DB	ahaxlist	AHA Database (excluding paced records)
ESC DB	esclist	European ST-T Database
NST DB	nstlist	Noise Stress Test Database
CU DB	culist	Creighton University Sustained Ventricular Arrhythmia Database

SEE ALSO

bxb(1), **epicmp**(1), **mxm**(1), **plotstm**(1), **rxr**(1), **setwfdb**(1), **sumstats**(1)
Evaluating ECG Analyzers

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/ecgeval.c

NAME

ecgpuwave – QRS detector and waveform limit locator

SYNOPSIS

ecgpuwave -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

ecgpuwave analyses an ECG signal from the specified *record*, detecting the QRS complexes and locating the beginning, peak, and end of the P, QRS, and ST-T waveforms. The output of **ecgpuwave** is written as a standard WFDB-format annotation file associated with the specified *annotator*. This file can be converted into text format using **rdann**(1) or viewed using **wave**(1).

The QRS detector is based on the algorithm of Pan and Tompkins (reference 1) with some improvements that make use of slope information (reference 2). Optionally, QRS annotations can be provided as input (see option **-i**), permitting the use of external QRS detectors such as **sqrs**(1) or manually-edited annotations (which can be created using **wave**(1)). The waveform limit locator is based on the algorithm described in reference 3 and evaluated in references 3 and 4.

The output annotation file contains PWAVE ("**p**") and TWAVE ("**t**") annotations that indicate the P- and T-wave peaks, as well as QRS annotations (NORMAL ("**N**") if generated by the built-in QRS detector, or copies of the input QRS annotations if these were supplied). **ecgpuwave** classifies each T wave as type 0 (normal), 1 (inverted), 2 (positive monophasic), 3 (negative monophasic), 4 (biphasic negative-positive), or 5 (biphasic positive-negative); this numeric classification is written into the **num** field of each TWAVE annotation. The P, QRS, and T waveform onsets and ends are marked in the output annotation file using WFON ("**(**") and WFOFF ("**)**") annotations. The **num** field of each WFON and WFOFF annotation designates the type of waveform with which it is associated: 0 for a P wave, 1 for a QRS complex, or 2 for a T wave.

Options include:

-f *time* Begin at the specified *time* (default: the beginning of the record).

-i *input-annotator*
Read QRS locations from the specified *input-annotator* (and copy them to the output annotation file). Default: run the built-in QRS detector.

-n *beat-type*
Specify which beats to process (must be used together with **-i**): *beat_type* may be 0 (default: process all beats) or 1 (process only beats labelled as NORMAL ("**N**") by the input annotator).

-s *n* Analyze signal *n* (default: signal 0).

-t *time* Stop at the specified *time* (default: the end of the record).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

rdann(1), **sqrs**(1), **wave**(1), **wqrs**(1)

REFERENCES

1. Pan J and Tompkins WJ. A Real-Time QRS Detection Algorithm. *IEEE Transactions on Biomedical Engineering* **32**(3):230-236, 1985.
2. Laguna P. *New Electrocardiographic Signal Processing Techniques: Application to Long-term Records.* Ph. D. dissertation, Science Faculty, University of Zaragoza, 1990.
3. Laguna P, Jané R, Caminal P. Automatic Detection of Wave Boundaries in Multilead ECG Signals: Validation with the CSE Database. *Computers and Biomedical Research* **27**(1):45-60, 1994.
4. Jané R, Blasi A, García J, and Laguna P. Evaluation of an automatic threshold based detector of waveform limits in Holter ECG with the QT database. *Computers in Cardiology* **24**:295-298 (1997; available at http://www.physionet.org/physiobank/database/qtdb/eval/)

AVAILABILITY

ecgpuwave is available as part of PhysioToolkit under the GPL (see **SOURCE** below).

AUTHORS

Pablo Laguna (laguna@posta.unizar.es), Raimon Jané, Eudald Bogatell, and David Vigo Anglada

SOURCE

http://www.physionet.org/physiotools/ecgpuwave/src/

NAME

edf2mit, mit2edf – convert between EDF and WFDB-compatible formats

SYNOPSIS

edf2mit -i *edffile* [*options ...*]
mit2edf -r *record* [*options ...*]

DESCRIPTION

These programs convert EDF (European Data Format) files into WFDB-compatible files (as used in PhysioBank) and vice versa. European Data Format was originally designed for storage of polysomnograms.

edf2mit reads the specified *edffile* and creates WFDB-compatible signal and header files containing the same data. Options for **edf2mit** include:

-b Input is in big-endian byte order (default: little-endian).

-h Print a brief usage summary.

-r *record*
Create the specified *record* (default: use the patient ID field from the input file as the record name).

-s *signal-list*
Copy only the signals named in the *signal-list* (one or more input signal numbers, separated by spaces; default: copy all signals). Signals are numbered consecutively beginning with zero. This option may be used to re-order or duplicate signals.

-v Verbose mode (print debugging output).

mit2edf reads the specified WFDB-format *record* (header and signal files) and creates an EDF file containing the same data. Output from **mit2edf** is always in the standard little-endian format. Options for **mit2edf** include:

-h Print a brief usage summary.

-o *file* Write output to the specified *file* (default: *record***.edf**).

-v Verbose mode (print debugging output).

Note that WFDB format does not include a standard way to specify the transducer type or the prefiltering specification; these parameters are not preserved by these conversion programs. Also note that use of the standard signal and unit names specified for EDF is permitted but not enforced by **mit2edf**.

Many EDF files contain signals at widely varying sampling frequencies. **edf2mit** handles these properly, but the default behavior of most WFDB applications is to read such data in low-resolution mode (in which all signals are resampled at the lowest sampling frequency used for any signal in the record). This is almost certainly not what you want if, for example, the record contains EEG signals sampled at 200 Hz and body temperature sampled at 1 Hz; by default, applications such as **rdsamp** and **wave** will resample the EEGs (and any other signals in the record) at 1 Hz. To avoid this behavior, you can use the **-H** (high resolution) option provided by **rdsamp**, **wave**, and a few other WFDB applications, or you can set the environment variable **WFDBGVMODE** to 1 (or any non-zero value) to specify that signals are to be read in high-resolution mode (in which all signals are resampled at the highest frequency used for any signal in the record). Setting **WFDBGVMODE** works with all WFDB applications, not only those that support the **-H** option. For further information, see the section titled "Multi-Frequency Records" in chapter 5 of the *WFDB Programmer's Guide*.

Note that applications built using version 10.4.5 and later versions of the WFDB library can read EDF files directly, so that the conversion performed by **edf2mit** is no longer necessary. The native WFDB files produced by **edf2mit** can be read more efficiently and with lower latency and memory requirements than the EDF files; in most cases, however, the difference will not be noticeable.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

AVAILABILITY

These programs are provided in the *convert* directory of the WFDB Software Package. Run **make** in that directory to compile and install them if they have not been installed already.

The PhysioNet ATM (http://physionet.org/cgi-bin/ATM) provides web access to **mit2edf** (select **Export signals as EDF** from the Toolbox).

SEE ALSO

a2m(1), **rdedfann**(1), **snip**(1), **xform**(1), **wfdb**(3), **header**(5)

Bob Kemp, Alpo Värri, Agostinho C. Rosa, Kim D. Nielsen and John Gade. A simple format for exchange of digitized polygraphic recordings. *Electroencephalography and Clinical Neurophysiology* **82**:391-393 (1992).

Bob Kemp's EDF web site (http://www.edfplus.info/). The definitive reference on the format; it includes the full specification of EDF from the 1992 paper, sample EDF files, software for reading and viewing them, FAQs, and much more.

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/convert/edf2mit.c
http://www.physionet.org/physiotools/wfdb/convert/mit2edf.c

NAME

edr – derive a respiration signal from an ECG

SYNOPSIS

edr -r *record* **-i** *annotator* [*options ...*]

DESCRIPTION

edr derives a sample of a respiratory signal for each QRS complex in the input ECG, by measuring the mean electrical axis (in two-channel mode) or the projection of that axis onto the lead axis (in single-channel mode). See the references below for details of the algorithm.

edr reads the signal and annotation files specified by *record* and *annotator*, and writes another annotation file, which is a copy of the input annotation file except that the **num** field of each beat annotation is replaced by an EDR sample.

If the beat annotations are not located at the QRS peaks, it will be necessary to set the window limits (the offsets relative to the annotations between which the raw measurements for the EDR are taken), using the **-d** option. By default, **edr** behaves as if the option **-d -0.04 0.04** has been given (in other words, measurements are taken over an 80 ms window beginning 40 ms (.04 seconds) before the annotation, and ending 40 ms after the annotation); this default is reasonable if the QRS annotations have been placed on or near the QRS peaks or centroids. If **edr** is supplied with annotations generated by **sqrs**, or another method that places the annotations near the PQ junction (the beginning of the QRS complex), the option **-d 0 0.08** is recommended.

For ECGs sampled at relatively low rates (e.g., 100-128 Hz, as is common for many long-term ECG recordings), it may be advantageous to base the EDR on the T-wave rather than the QRS complex, by choosing a window such as **-d -0.08 0.28** or **-d -0.12 0.32** (for annotations placed at the QRS peaks or PQ junctions respectively), since this permits an axis estimation based on a larger number of samples. Note that the use of a negative value for *dt1*, as in these examples, allows the beginning of the EDR measurement window to be placed *after* the QRS annotation.

Options include:

-d *dt1 dt2*
: Set the EDR measurement window relative to QRS annotations (defaults: *dt1* = 0.04 (seconds before annotation), *dt2* = 0.04 (seconds after annotation).

-f *time* Begin at the specified *time* (default: the beginning of the record).

-h Print a usage summary.

-o *ann* Use *ann* as the output annotator name (default: **edr**).

-s *signal-list*
: Analyze only the signals named in the *signal-list* (one or more input signal numbers, separated by spaces; default: analyze signals 0 and 1). If the *signal-list* contains more than two signals, only the first two are analyzed.

-t *time* Stop at the specified *time*.

-v Verbose mode: print individual measurements.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

Example

edr -r 100 -i atr -f 0 -t 5:0

This command creates an annotation file named **edr.100**, containing a copy of the reference (**atr**) annotation file for the first five minutes of record **100**, with EDR measurements for each annotated beat in the **num** fields of the output annotation file.

AVAILABILITY

edr is available as part of PhysioToolkit under the GPL (see **SOURCE** below).

SEE ALSO

plt(1), rdann(1), **setwfdb**(1)

Moody GB, Mark RG, Zoccola A, Mantero S. Derivation of respiratory signals from multi-lead ECGs. *Computers in Cardiology* **12**:113-116 (1985; available at http://www.physionet.org/physiotools/edr/cic85/)

Moody GB, Mark RG, Bump MA, et al. Clinical validation of the ECG-derived respiration (EDR) technique. *Computers in Cardiology* **13**:507-510 (1986; available at http://www.physionet.org/physiotools/edr/cic86/)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/edr/edr.c

NAME

epicmp – ANSI/AAMI-standard episode-by-episode annotation comparator

SYNOPSIS

epicmp -r *record* **-a** *reference-annotator test-annotator* [*options ...*]

DESCRIPTION

This program implements the VF, AF, and ST episode-by-episode comparison algorithms specified by the current American National Standard for ambulatory ECG analyzers (ANSI/AAMI EC38:2007). **epicmp** is the reference implementation of these algorithms, and must be used to obtain the episode-by-episode performance statistics cited in EC38 in order to be in compliance with the standard (see EC38, section 5.2.14).

Input to this program consists of two annotation files associated with the same *record*. One of these is designated the *reference* annotation file, the other the *test* annotation file.

Options include:

-A *file* Append atrial fibrillation detection reports to the specified *file*.

-f *time* Begin the comparison at the specified *time* (default: 5 minutes after the beginning of the record).

-h Print a usage summary.

-i *time* Exclude episodes shorter than *time* (default: 0 seconds) from episode statistics.

-I *time* Exclude episodes shorter than *time* (default: 0 seconds) from episode and duration statistics. (At most one of **-i** and **-I** may be used.)

-l Write reports in line format (default: matrix format).

-L Same as **-l**.

-S *file1 file2*
Append ischemic ST episode detection reports to *file1*, and ST deviation measurements to *file2*.

-S0 *file1 file2*
As for **-S**, but report on signal 0 only.

-S1 *file1 file2*
As for **-S**, but report on signal 1 only.

-t *time* Stop the comparison at the specified *time* (default: the end of the record if it is defined, the end of the reference annotation file otherwise; if *time* is 0, the comparison ends when the end of either annotation file is reached).

-V Append ventricular flutter and fibrillation detection reports to the specified *file*.

-x Exclude periods of atrial fibrillation from calculations of atrial fibrillation positive predictivity, as required by EC38:1998 (default: include these periods, as required by EC38:2007).

The episode and duration statistics gathered by **epicmp** are based on tallies of overlapping episodes in the reference and test annotation files. Duration statistics give weight to each episode or detection in proportion to its duration. Episode statistics give equal weight to each episode or detection, irrespective of length; each test-annotated episode that meets the criteria for overlap (see below) with a reference-annotated episode is counted as a true positive. Episodes are defined as follows (see *<wfdb/ecgcodes.h>* for definitions of annotation types):

Atrial fibrillation episodes
begin with a **RHYTHM** annotation, with the *aux* field containing the text '**(AFIB**', and end with any other **RHYTHM** annotation (or at the end of the record). Reference-marked episodes of atrial flutter (begun by **RHYTHM** annotations with the text '**(AFL**') are excluded from AF comparisons (i.e., the test annotator is neither penalized nor rewarded for its treatment of atrial flutter in this context). Any amount of overlap is sufficient to qualify a test episode as a true positive.

Ventricular fibrillation or flutter episodes
begin with a **VFON** annotation, and end with a **VFOFF** annotation (or at the end of the record). **RHYTHM** annotations are ignored in this context by **epicmp**. Any amount of overlap is sufficient to qualify a test episode as a true positive.

Ischemic ST episodes
begin with a **STCH** annotation, with the *aux* field containing the text '**(ST***ns*', and end with another **STCH** annotation, with the text '**ST***ns***)**' (or at the end of the record). Between these annotations, the extremum (the time at which the absolute value of the ST deviation is greatest) is marked with another **STCH** annotation, with the text '**AST***nsm*'; this annotation may be omitted in the test annotation file. In these annotations, *n* is '**0**' or '**1**', and denotes the affected signal; *s* is '**+**' for episodes of ST elevation, or '**-**' for episodes of ST depression; and *m* is the ST deviation in microvolts, relative to a reference level established from the first 30 seconds of the record. The values of *s* and *m* are not significant for the episode comparison made by *epicmp*. When using the **-S0** or **-S1** options, *n* must be 0 or 1 respectively; other **STCH** annotations are ignored. When using the **-S** option, the value of *n* is ignored: each '**(ST***ns*' annotation increments a counter, and each '**ST***ns***)**' annotation decrements the counter; in this context, ST episodes begin when the counter becomes positive and end when the counter reaches zero (or at the end of the record). To qualify a test episode as a true positive for purposes of determining ST episode sensitivity, it must overlap at least 50% of the reference episode, or the overlap must include the reference-marked extremum. To qualify a test episode as a true positive for purposes of determining ST episode positive predictivity, the reference episode must overlap at least 50% of the test episode, or the overlap must include the test-marked extremum, if present.

The second file generated when using the '**-S**', '**-S0**', or '**-S1**' options contains comparisons of ST deviation measurements wherever such measurements are available in the reference annotation files. In the existing databases, these appear only at extrema within each annotated ischemic (or non-ischemic) ST episode, as described above. For purposes of comparison of ST deviation measurements, test ST measurements for each signal are read from the *aux* field of beat annotations, which should contain text of the format '*m n*' (where *m* and *n* are the measured ST deviations for signals 0 and 1 respectively). If these measurements are missing from any test beat annotation, **epicmp** assumes that they have not changed since they last appeared. **epicmp** ignores '**AST**...' annotations in the test annotation file when making this comparison. In the output file, any test measurements that deviate from the reference measurements by more than 100 microvolts are tagged with an asterisk ('*****'). **plotstm**(1) can produce a scatter plot of these data using this file as input.

At least one of the options '**-A**', '**-S**', '**-S0**', '**-S1**', and '**-V**' must be used. If '**-**' is given as a *file* argument, reports are written on the standard output. The output generated by selecting **-l** or **-L** includes column headings only if a *file* other than '**-**' is specified, and only if the specified *file* does not already exist. In this way, **epicmp** can be used repeatedly to build up line-format tables for multiple records, for further processing by **sumstats**(1).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

DIAGNOSTICS

non-standard comparison selected
The **-f**, **-i**, **-I**, and **-t** options modify the comparison algorithms used by **epicmp** in ways not permitted by EC38. These options are provided for the use of developers, who may find them useful for obtaining a more detailed understanding of algorithm errors.

BUGS

Since **epicmp** performs multiple passes over its input files, it cannot be used at the end of a pipe.

Between 1992 and 2002, this program was known as **epic**; the name was changed to avoid conflict with a new but widely distributed IRC chat client also named **epic**. By analogy to **bxb**, **mxm**, and **rxr**, this program should have been called **exe**, which would have created interesting possibilities for confusion.

SEE ALSO

bxb(1), ecgeval(1), mxm(1), plotstm(1), rxr(1), setwfdb(1), sumstats(1)
Evaluating ECG Analyzers (in the *WFDB Applications Guide*)
American National Standard ANSI/AAMI EC38:1998, Ambulatory Electrocardiographs; available from AAMI, 1110 N Glebe Road, Suite 220, Arlington, VA 22201 USA (http://www.aami.org/).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/epicmp.c

NAME

fft – fast Fourier transform

SYNOPSIS

fft [*options ...*] *input-file*

DESCRIPTION

fft transforms a real-valued time series (from the specified *input-file*, or from the standard input if *input-file* is specified as "-"; *input-file* must be in text form) into a frequency spectrum (on the standard output). Using appropriate options, **fft** can produce polar or rectangular format amplitude spectra, or power spectra, or it can perform an inverse FFT to transform a polar or rectangular format amplitude spectrum into a time series. The input series may be corrected if it has a non-zero mean amplitude or first derivative (by 'zero-meaning' or 'detrending' the input series). Output spectra may be smoothed in several different ways.

By default, the standard output is the magnitude of the discrete Fourier transform of the input series, normalized such that the mean of the squares of the inputs is equal to the sum of the squares of the outputs (i.e., the RMS power determined from the time series equals the total power determined from the spectrum; this normalization is correct only if the input series has a mean value of zero).

Options are:

-c Output unnormalized complex FFT (real components in first column, imaginary components in second column).

-f *frequency*
Show the center frequency for each bin in the first column. The *frequency* argument specifies the input sampling frequency; the center frequencies are given in the same units.

-h Print a usage summary.

-i Perform inverse FFT; in this case, the standard input should be in the form generated by **fft -c**, and the standard output is a series of samples. No other options may be used with **-i**.

-I Perform inverse FFT as above, but using input generated by **fft -p**. No other options may be used with **-I**.

-l *n* Perform up to *n*-point transforms. **fft** rounds *n* up to the next higher power of two unless *n* is already a power of two. If the input series contains fewer than *n* samples, it is padded with zeros up to the next higher power of two. Any additional input samples beyond the first *n* are not read. Default: *n* = 16384.

-n *n* Process the input in overlapping chunks of *n* samples and output an averaged spectrum. If used in combination with **-P**, the output is the average of the individual squared magnitudes; otherwise, the output is derived from the averages of the real components and of the imaginary components taken separately. For best results, *n* should be a power of two.

-N *n* Process the input in overlapping chunks of *n* samples and output a spectrum for each chunk. Successive spectra are concatenated in the output. Only one of **-n** and **-N** may be used at a time. For best results, *n* should be a power of two.

-p Show the phase in radians in the last column.

-P Generate a power spectrum (print squared magnitudes).

-s *n* Smooth the output by applying an *n*-point moving average to each bin. This option does not change the number of bins.

-S *n* Smooth the output by summing sets of *n* consecutive bins. This option reduces the number of bins by a factor of *n*.

-w *window-type*
Apply the specified window to the input data. *window-type* may be one of: 'Bartlett', 'Blackman', 'Blackman-Harris', 'Hamming', 'Hanning', 'Parzen', 'Square', and 'Welch'. The 'Square' window type is equivalent to using no window at all; this is also variously known as a rectangular or Dirichlet window.

-z Add a constant to each input sample, chosen such that the mean value of the entire series is zero.

-Z Set the mean value of the inputs to zero as for **-z**, and detrend the series (set its mean first derivative to zero). This is equivalent to subtracting a best-fit (by least squares) line from the input data.

BUGS

Because of accumulated round-off errors, the command

fft -p <*file1* **| fft -I >***file2*

may not produce an exact copy of *file1* in *file2*, even if the number of samples is an exact power of 2. Using rectangular form, as in the command

fft -c <*file1* **| fft -i >***file2*

produces smaller errors, and is slightly faster than using polar form as in the first example.

SEE ALSO

coherence(1), **hrfft**(1), **lomb**(1), **memse**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/psd/fft.c

NAME

fir – general-purpose FIR filter for WFDB records

SYNOPSIS

fir [*options ...*] **-c** [*coefficients ...*]

DESCRIPTION

fir can be used to apply any desired finite impulse response filter to any desired section of a waveform database record. *Options* are:

-c *coefficient* [*coefficient ...*]
Filter using the specified *coefficients* (must be the last option; **-c** marks the beginning of the coefficient list).

-C *file* Read the filter coefficients from the specified file rather than from the argument list.

-f *time* Filter from the specified *time* on the input record (default: start at the beginning of the record).

-h Print a usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-i *record*
Use the specified *record* for input (default: record 16).

-n *record*
Create a header file for the output signals, with the specified *record* name. The signal descriptions are copied from those of the input signals.

-o *record*
Use the specified *record* for output (default: record 16).

-ri Rectify the input (i.e., take its absolute value) before filtering.

-ro Rectify the filtered output.

-s *shift* To compensate for phase shift, read ahead on the input record by the specified interval before starting the filter. *Shift* is specified in standard time format (use **s***nn* to compensate for a phase shift of *nn* samples).

-t *time* Filter until the specified *time* on the input record (default: go to the end of the record).

Unless the **-C** option is used, the **-c** argument should appear at the end of the option list. Filter coefficients are real numbers separated by spaces; the last coefficient is applied to the most recent input sample.

In the present implementation, the same filter is applied to each input signal. If the output record header file specifies fewer signals than are present in the input, any extra input signals are discarded.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

Examples

A low-pass "boxcar" filter:
fir -c .2 .2 .2 .2 .2

The complementary high-pass filter:
fir -c -.2 -.2 .8 -.2 -.2

An attenuator:
fir -c .4

A differentiator:
fir -c -1 1

A 60-Hz notch filter, with partial correction for phase shift, for the MIT–BIH database (360

samples/second):

fir -s s2 -c .5 0 0 .5

A "triangle" filter for QRS detection (at 128 samples/second):

fir -s s8 -c -1 -2 -3 -4 -1 2 5 8 5 2 -1 -4 -3 -2 -1

SEE ALSO

mfilt(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/fir.c

NAME

gqfuse – combine QRS annotation files

SYNOPSIS

gqfuse -r *record* -a *annotator1 annotator2 ...* [*options ...*]

DESCRIPTION

gqfuse produces a QRS annotation file based on two or more input QRS annotation files with annotator names *annotator1*, *annotator2*, etc. Each one-minute segment of the output annotation file is a copy of the corresponding segment of one of the input annotation files. In each segment, the program copies the input that best matches a predicted heart rate. If there are N inputs, the prediction is the median of N+1 values (the previous prediction and the number of beats marked within the current segment of each of the N input files). Although this process allows the input to be switched once per minute, the policy for resolving ties (within 2 beats) favors not switching if the previously chosen input is one of those belonging to the tie.

As its name suggests, **gqfuse** is intended to be used as a companion to the **gqrs**(1) QRS detector, but it is able to process annotations from any beat detector. Non-beat annotations (e.g., rhythm, signal quality, artifact, non-QRS waveforms, and notes) are copied to the output if present in the best matching input segments, but they are not counted as beats by **gqfuse** when it makes heart rate predictions.

One way to use **gqfuse** is to combine input annotation files for each available ECG signal in a record, made using a single detector such as **gqrs**. Another is to combine input annotation files made using a variety of QRS detectors. These ideas can be combined as desired.

A configuration file, which can be shared with **gqrs** and **gqpost**(1), can be used to specify the expected heart rate. (In future versions, other parameters in the configuration file may also be used by **gqfuse**). The configuration file is unnecessary when processing adult human ECGs, but an appropriately constructed configuration file allows **gqrs** to analyze fetal, pediatric, and animal ECGs.

Options include:

-c *file* Initialize parameters based on the specified (text) configuration *file*. See the example configuration file, *gqrs.conf*, for details.

-h Print a usage summary.

-o *name*
Write annotations to an annotation file with the specified annotator *name* (default: **gqf**).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

gqrs(1), **ecgpuwave**(1), **setwfdb**(1), **sqrs**(1), **wqrs**(1)

AUTHORS

George B. Moody (george@mit.edu).

SOURCES

http://www.physionet.org/physiotools/wfdb/app/gqfuse.c
http://www.physionet.org/physiotools/wfdb/app/gqrs.conf

NAME

gqrs, gqpost – QRS detector and post-processor

SYNOPSIS

gqrs -r *record* [*options ...*]
gqpost -r *record* [*options ...*]

DESCRIPTION

gqrs attempts to locate QRS complexes in an ECG signal in the specified *record*. The detector algorithm is new and as yet unpublished. The output of **gqrs** is an annotation file (with annotator name **qrs**) in which all detected beats are labelled normal ("N"). The *subtyp*, *chan*, and *num* fields of each annotation respectively indicate the detection pass (0 or 1) during which the QRS complex was detected, the signal number on which it was detected, and the peak amplitude of the detector's matched filter during the QRS complex.

As a QRS detector for research, **gqrs** has been optimized for sensitivity. **gqpost** can post-process **gqrs**'s output annotation file to improve positive predictivity, generally at a cost of reduced sensitivity. It does this by copying its input annotation file, changing N annotations into artifact ("|") annotations if they are likely to be erroneous.

A configuration file shared by **gqrs** and **gqpost** can be used to describe some of the expected characteristics of the ECG signal. This is unnecessary when processing adult human ECGs, but an appropriately constructed configuration file allows **gqrs** to analyze fetal, pediatric, and animal ECGs. A sample configuration file is available (see SOURCES, below); it contains details about all configurable parameters.

Options include:

-a *annotator*
: [gqpost only] Read annotations from the specified *annotator* (default: qrs).

-c *file* Initialize parameters based on the specified (text) configuration *file*. See the example configuration file, *gqrs.conf*, for details.

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a usage summary.

-H Read the signal files in high-resolution mode (default: standard mode).

-m *threshold*
: Specify the *threshold* (default: 1.0) for detection [qqrs] or acceptance [gqpost]. Use higher values to reduce false detections, or lower values to reduce the number of missed beats.

-n *name*
: [gqrs only] Save the filtered signals in a new record with the specified record *name*.

-o *name*
: [gqpost only] write annotations to an annotation file with the specified annotator *name*.

-s *signal*
: [gqrs only] Specify the *signal* to be used for QRS detection (default: 0). Note that signals may be specified by number or name.

-t *time* Process until the specified *time* in *record* (default: the end of the *record*).

Note that **gqpost** always copies its entire input annotation file. The **-f** and **-t** options, if present, only define the interval during which **gqpost** may change annotations. Since **gqpost** can reprocess its own output, this feature allows multiple passes using different threshold values and processing intervals, if necessary.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

EXAMPLES

To mark QRS complexes in record 100 beginning 5 minutes from the start, ending 10 minutes and 35 seconds from the start, and using signal 1, use the command:

gqrs -r 100 -f 5:0 -t 10:35 -s 1

The output annotations may be read using (for example):

rdann -a qrs -r 100

To evaluate the performance of this program, run it on the entire record, by:

gqrs -r 100

and then compare its output with the reference annotations by:

bxb -r 100 -a atr qrs

SEE ALSO

bxb(1), **ecgpuwave**(1), **rdann**(1), **setwfdb**(1), **sqrs**(1), **wqrs**(1), **xform**(1)

AUTHORS

George B. Moody (george@mit.edu).

SOURCES

http://www.physionet.org/physiotools/wfdb/app/gqrs.c
http://www.physionet.org/physiotools/wfdb/app/gqpost.c
http://www.physionet.org/physiotools/wfdb/app/gqrs.conf (sample configuration file)

NAME

hrfft, hrlomb, hrmem – calculate and plot heart rate power spectra
hrplot – plot heart rate time series

SYNOPSIS

hrfft [*options ...*]
hrlomb [*options ...*]
hrmem [*options ...*]
hrplot [*options ...*]

DESCRIPTION

The first three of these UNIX shell scripts are intended to illustrate the use of **fft**(1), **lomb**(1), and **memse**(1) by producing heart rate power spectra using the fast Fourier transform, the Lomb periodogram, and the maximum entropy (all poles) method (also known as autoregressive, or AR, power spectral density estimation). All four programs derive heart rate time series from beat annotation files. **hrfft** and **hrmem** use **tach**(1) to obtain a uniformly resampled heart rate time series from the annotation file, which is then used as input to **fft** or **memse**, and the spectrum thereby obtained is then plotted. **hrlomb** and **hrplot** use **ihr**(1) to obtain an irregularly sampled heart rate time series. **hrplot** plots this time series directly, and **hrlomb** uses it as input to **lomb**, and then plots the spectrum.

All four programs accept the same *options*:

-a *annotator*
: Read annotations from the specified *annotator* (default: the value of the environment variable **ANNOTATOR**, if set).

-f *time* Begin at the specified *time* within the annotation file (default: the value of the environment variable **START**, if set, or the beginning of the file otherwise).

-l *axes* Log-transform the specified *axes* (default: use linear axes). The *axes* can be specified as **x**, **y**, or **xy**.

-p *plot-utility*
: Use the specified *plot-utility* to generate the output (default: the value of the environment variable **PLOT**, if set, or **plt**(1), if it exists, or **plot2d**(1) otherwise).

-r *record*
: Produce a heart rate power spectrum for the specified *record* (default: the value of the environment variable **RECORD**, if set).

-t *time* Stop at the specified *time* within the annotation file (default: the value of the environment variable **END**, if set, or the end of the file otherwise).

-T *device*
: Produce output on the specified *device* (default: the screen). The *device* must be among those supported by the *plot-utility* (see above).

If *annotator* or *record* are not specified using environment variables or command-line options, these programs obtain values from the user interactively.

Although **hrfft**, **hrlomb**, and **hrmem** all produce power spectra, the units of power differ among them. Absolute comparisons can be made only between spectra produced using the same method, from time series of the same length.

Note that these shell scripts can be run under MS-DOS using a suitable set of UNIX-like utilities, such as the MKS Toolkit or the GNUish MS-DOS utilities, and under MS-Windows using the free Cygwin package.

ENVIRONMENT

In addition to the variables **ANNOTATOR**, **END**, **PLOT**, **RECORD**, and **START**, it may be necessary to set **WFDB** (see **setwfdb**(1)).

SEE ALSO

fft(1), **ihr**(1), **lomb**(1), **memse**(1), **plot2d**(1), **plt**(1), **setwfdb**(1), **tach**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/psd/hrfft
http://www.physionet.org/physiotools/wfdb/psd/hrlomb
http://www.physionet.org/physiotools/wfdb/psd/hrmem
http://www.physionet.org/physiotools/wfdb/psd/hrplot

NAME

hrstats – collect and summarize heart rate statistics from an annotation file

SYNOPSIS

hrstats -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

hrstats reads the annotation file specified by *record* and *annotator* and produces a uniformly sampled and smoothed instantaneous heart rate signal, using the IPFM model as originally implemented in tach.c. In this context, heart rate (HR) is defined as ten times the number of beat-to-beat (RR) intervals (and fractional intervals) within a 6-second HR measurement window, including intervals beginning and/or ending with ectopic beats, and RR intervals are defined by the locations of consecutive beat annotations in the annotation file.

The first HR window starts at the beginning of the first RR interval in the record and ends 6 seconds later. Subsequent HR windows begin at 1-second intervals following the first, i.e, they overlap by 5/6 (83.33%). HR windows that contain part or all of a very long (>3 sec) or very short (<0.2 sec) interval are discarded. All others are used to generate HR measurements that are accumulated in a histogram with 1 bpm bins.

When all intervals have been processed, summary statistics calculated from the histogram are written to the standard output and to a WFDB '.info' file, in this format:

<HR>: 71|73/75/81|86 +-1.8 bpm [atr]

This example is the output of 'hrstats -r mitdb/100 -a atr'. From left to right, it shows:

the characteristic that is summarized [HR]
extreme low value [in the example, 71]
5th percentile [73]
mean (trimmed, excluding outliers below 5th or above 95th percentiles) [75]
95th percentile [81]
extreme high value [86]
sample deviation (of values included in the trimmed mean) [1.8]
units of all statistics [bpm]
source of data [atr annotations]

Optionally, the HR histogram can also be written to a file containing two columns separated by a tab. The second column contains the number of measurements of heart rate falling within 0.5 bpm of the heart rate in the first column. The histogram includes only heart rates between the extreme low and extreme high heart rates inclusive.

Options include:

-h Print a usage summary.

-o *file* Write the HR histogram to the specified *file*.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

ihr(1), **tach**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/hrstats.c

NAME

ihr – calculate instantaneous heart rate

SYNOPSIS

ihr -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

ihr reads an annotation file (specified by the *annotator* and *record* arguments) and produces an instantaneous heart rate signal (from the reciprocals of the interbeat intervals.) Unlike **tach**(1), however, **ihr** does not resample its output in order to obtain uniform time intervals between output samples. (If there is any variation whatsoever in heart rate, the intervals between output samples will be non-uniform.) This property makes the output of **ihr** unsuitable for conventional power spectral density estimation, but ideal for PSD estimation using the Lomb periodogram (see **lomb**(1)).

Options include:

-d *tolerance*
: Reject beat-to-beat heart rate changes exceeding *tolerance* (in beats per minute; default: 10). Any intervals for which the calculated heart rate would differ by more than the specified tolerance are simply excluded from the output series. To disable this behavior, use a large value for *tolerance* (e.g., 10000).

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a usage summary.

-i Include all intervals bounded by QRS annotations (default: include intervals bounded by consecutive supraventricular beats only).

-p *type ...*
: Include intervals bounded by annotations of the specified *types* only. The *type* arguments should be annotation mnemonics (e.g., **N**) as normally printed by **rdann**(1) in the third column. More than one **-p** option may be used in a single command, and each **-p** option may have more than one *type* argument following it. If *type* begins with "-", however, it must immediately follow **-p** (standard annotation mnemonics do not begin with "-", but modification labels in an annotation file may define such mnemonics).

-t *time* Process until the specified *time* in *record* (default: the end of the *record*).

-v Print the output sample number before each output sample value.

-v, **-vs**, **-vm**, **-vh**, **-V**, **-Vs**, **-Vm**, **-Vh**
: Print the elapsed times from the beginning of the record to the annotations that begin each interval, as sample number (using **-v**), or in seconds (using **-vs**), minutes (using **-vm**), or hours (using **-vh**) before each heart rate value. The options **-V**, **-Vs**, **-Vm**, and **-Vh** work in the same way, but the printed times are those for the annotations that end the intervals. Only one of these options can be used at a time; if none is chosen, **-vs** mode is used by default.

-x Exclude the interval immediately following each rejected interval. (Rejected intervals are those bounded by excluded beats on at least one end, and those that do not satisfy the *tolerance* criterion). By default, intervals following rejected intervals are included (unless they are rejected by the *tolerance* criterion), and a third column is used to flag these intervals (a zero in the third column means the interval is normal, a one means it follows an excluded interval).

Reference ('atr') annotation files can be used as input to **ihr**, but files that contain manually-inserted annotations are less suitable, since annotation placement is likely to be less consistent than in annotation files generated by programs such as **sqrs**(1).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

lomb(1), **setwfdb**(1), **sqrs**(1), **tach**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/ihr.c

NAME

imageplt – plot a greyscale image

SYNOPSIS

imageplt -d *nrows ncols* [*options ...*] [*file*]

DESCRIPTION

imageplt provides a simple way to plot a greyscale image using **plt**(1). The required arguments, *nrows* and *ncols*, specify the numbers of rows and columns in the image. The input *file* (or the standard input, if no input file is specified) contains only the grey levels for each pixel (0 = white, 1 = black). Each entry is an ASCII-coded decimal floating point number, separated from adjacent entries by whitespace (one or more spaces, tabs, or newlines). The first *nrows* entries are the grey levels for column 0 of the image, botttom to top, and each successive column from left to right of the image follows. If *nrows* is small, it may be convenient to arrange the image file in columns and rows corresponding to those of the image, but this is not necessary. In no case should the length of a line of input exceed 50000 bytes (defined as MAXLEN in the source).

Options include:

-n Generate a negative image (1 = white, 0 = black).

-x *xmin xmax*
Specify the range of the x-coordinates (default: *xmin*=0, *xmax*=*nrows*-1).

-y *ymin ymax*
Specify the range of the y-coordinates (default: *ymin*=0, *ymax*=*ncols*-1).

The output of *imageplt* is text in three columns, to be plotted using the **-pc** option of **plt**, as in:

```
imageplt -d 10 10 foo | plt 0 1 2 -pc
```

SEE ALSO

plt(1), **pltf**(1)

AVAILABILITY

imageplt is available as part of the **plt** package in PhysioToolkit (see **SOURCES** below) under the GPL.

AUTHOR

George B. Moody (**george@mit.edu**)

SOURCE

http://www.physionet.org/physiotools/plt/plt/misc/imageplt.c

NAME

log10 – calculate common logarithms of two-column data

SYNOPSIS

log10

DESCRIPTION

log10 reads its standard input, which should be in text form and should contain two positive numbers (*x* and *y*) on each line, separated by spaces or tabs. The standard output of **log10** contains four columns of numbers, separated by spaces: the common (base 10) logarithms of *x* and *y*, and the *x* and *y* values. To avoid underflow, if any input is less than **MINDOUBLE** (defined in *<values.h>* as the smallest positive value that can be represented as a double-precision floating point quantity), it is replaced by that value.

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/psd/log10.c

NAME

lomb – estimate power spectrum using the Lomb periodogram method

SYNOPSIS

lomb [*options ...*] *input-file*

DESCRIPTION

lomb transforms a real-valued time series (from the specified *input-file*, or from the standard input if *input-file* is specified as "-"; *input-file* must be in text form) into a power spectrum (on the standard output), using a technique known as the Lomb periodogram.

The input is a text file containing a sampled time series, presented as two columns of numbers (the sample times and the sample values). The intervals between consecutive samples need not be uniform (in fact, this is the most significant advantage of the Lomb periodogram over other methods of power spectral density estimation). **lomb** writes the Lomb periodogram (the power spectral density estimate derived from the input time series) on the standard output, in two columns (frequency and power). If the units of the sample times in the input file are seconds, the units of the frequencies in the output are Hz.

Options are:

-h Print a usage summary.

-P Generate a power spectrum (print squared magnitudes).

-s Smooth the output.

-z Add a constant to each input sample, chosen such that the mean value of the entire series is zero.

Among many other applications, this program can be used to estimate heart rate power spectra, in combination with **ihr**(1). The Lomb method is ideal for analysis of any time series with missing or noisy data (the noisy data may be removed from the time series and need not be replaced, as would be necessary if conventional PSD estimation algorithms were employed).

SEE ALSO

fft(1), **hrfft**(1), **memse**(1)

Lomb, N.R. Least-squares frequency analysis of unequally spaced data. *Astrophysics and Space Science* **39**:447-462 (1976).

Press, W.H, and Rybicki, G.B. Fast algorithm for spectral analysis of unevenly sampled data. *Astrophysical J.* **338**:277-280 (1989).

Press, W.H. Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. *Numerical Recipes in C: the Art of Scientific Computing*, pp. 575-584 (Cambridge Univ. Press, 1992).

Moody, G.B. Spectral analysis of heart rate without resampling. *Computers in Cardiology 1993*, pp. 715-718 (IEEE Computer Society Press, 1993). http://www.physionet.org/physiotools/lomb/lomb.html .

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/psd/lomb.c

NAME

lwcat – postprocess output of plt to make PostScript, EPS, PDF or PNG

SYNOPSIS

plt -T lw ... **| lwcat** [*options* ...]

DESCRIPTION

lwcat collects the PostScript output of **plt**(1) and adds a prolog and epilog to create a complete PostScript document (or another format, if appropriate options have been selected). It is possible to concatenate the outputs of two or more **plt** runs to be processed as a single job by **lwcat**; see the *plt Tutorial and Cookbook* for details.

Output format

By default, **lwcat** sends its output directly to the default printer via **lpr**. These options may be used to modify this behavior:

-no Send the output to the printer, but don't eject the page (use this option if you wish to overlay the output with additional material to be produced by another program).

-ps Write PostScript to the standard output (not to the printer).

-psv Write PostScript to a temporary file and view it with **gv** (ghostscript).

-gv Same as **-psv**.

-eps Write EPSF (encapsulated PostScript format) to the standard output. Note that this is only a close approximation to EPSF; it is acceptable to LaTeX's epsfig package, at least.

-pdf Write PDF (portable document format) to the standard output.

-png Write PNG (portable network graphics) format to the standard output.

Window options

By default, the output appears within a 6.75x6 inch (171x152 mm) window, the lower left corner of which is positioned 1 inch (25.4 mm) from the left edge and 3.5 inches (89 mm) from the bottom edge of the page. The following options may be used to modify the size, location, and orientation of the output:

-landscape
Use landscape mode (rotate plot 90 degrees counterclockwise).

-sq Plot in a 6x6 inch (152x152 mm) square window, 1.25 inches (32 mm) from the left edge and 3.5 inches (89 mm) from the bottom edge of the page.

-t Plot in a 6.25x6.25 inch (159x159 mm) square window, positioned as for **-sq**.

-t2 Plot in a 6.25x4 inch (159x102 mm) window, positioned as for **-sq**.

-CinC Plot in a 4.75x3.15 inch (121x80 mm) window, positioned as for **-sq**.

-sq2 Plot in a 4.5x5.5 inch (114x140 mm) window, 2.5 inches (63 mm) from the left and bottom edges of the page.

-v Plot in a 7x9.5 inch (178x241 mm) window, 0.75 inches (19 mm) from the left and bottom edges of the page (centered on a US letter sheet).

-v2 Plot in a 7x8.5 inch (178x216 mm) window, positioned as for **-v**.

-va4 Plot in a 190x275 mm window, centered on an A4 sheet.

-full Plot in a 7.5x10 inch (191x254 mm) window, centered on a US letter sheet.

-slide Plot in a 7.5x5 inch (191x127 mm) window, 0.5 inches (12.7 mm) from the left edge and 3 inches (76 mm) from the bottom edge of the page (3:2 aspect ratio, as for 35 mm slides).

-screen Plot in a 7.5x5.625 inch (191x143 mm) window, 0.5 inches (12.7 mm) from the left edge and 2.375 inches (60 mm) from the bottom edge of the page (4:3 screen aspect ratio).

-golden
Plot in a 7.5x4.635 inch (191x118 mm) window, 0.5 inches (12.7 mm) from the left edge and 3.365 inches (85 mm) from the bottom edge of the page (the aspect ratio is approximately the "golden ratio", (1+sqrt(5))/2 = 1.61803 ...).

-strip Plot in an 8x0.8 inch (203x20 mm) window.

-custom *width height left-margin bottom-margin font_scale*
Plot in a custom window. The **-custom** option reads up to 5 arguments that follow it. If one of the 5 arguments immediately following **-custom** begins with '**-**', it and any remaining arguments are treated as ordinary options, and default values are used for any missing options. The units of width, height, and margins are inches, and the font scale is a dimensionless factor with a default value of 1 that can be used to enlarge or shrink any text in the plot. The defaults for *width* and *left-margin* are 5 and 0 respectively, and the defaults for *height* and *bottom-margin* are the values assigned to *width* and *left-margin* (whether explicitly or by default).

Other window options can be easily added; see the source for **lwcat** for details.

Copies
By default, **lwcat** prints a single copy. Multiple copies can be produced using the options **-c2**, **-c3**, **-c4**, **-c5**, and **-c6**; when using a PostScript printer, this will almost always be much faster than rerunning **lwcat**, since the document is downloaded and rasterized only once when using these options. To print more than 6 copies, repeat or combine these options as needed.

FILES
/usr/lib/ps/plt.pro
PostScript prolog

SEE ALSO
plt(1)

AVAILABILITY
lwcat is available as part of the **plt** package in PhysioToolkit (see **SOURCE** below) under the GPL.

AUTHORS
Paul Albrecht and George B. Moody (**george@mit.edu**)

SOURCE
http://www.physionet.org/physiotools/plt/src/lwcat

NAME

memse – estimate power spectrum using maximum entropy (all poles) method

SYNOPSIS

memse [*options ...*] *input-file*

DESCRIPTION

memse transforms a real-valued time series (from the specified *input-file*, or from the standard input if *input-file* is specified as "-"; *input-file* must be in text form) into a power spectrum (on the standard output). **memse** is designed to be used in much the same way as **fft**(1); it accepts the same input, produces output in the same format, and accepts many of the same options used with **fft**.

Unlike **fft**, which bases its spectral estimates on the discrete Fourier transform, **memse** uses the maximum entropy (all poles) method, also known as autoregressive (AR) spectral estimation. This method models the spectrum by a series expansion in which the free parameters are all in the denominators of its terms; hence each term may represent a pole (corresponding to infinite power spectral density within an infinitely narrow frequency band). By contrast, Fourier analysis models the spectrum by a series expansion in which the free parameters are all in the numerators; hence each term in a Fourier series may represent a zero. All-poles models are particularly useful for analysis of spectra which have discrete peaks (in the terminology of optical spectra, "lines").

In order to use **memse**, you should have some idea of the order of the model you wish to use (i.e., the number of poles). Although this may be any number up to the number of input points, the number of poles generally should not exceed the square root of the number of input points, and usually should be considerably less than that number. Large numbers of poles lead to lengthy computations (much slower than the FFT) in which accumulated roundoff error becomes a serious problem. This problem may also occur if the length of the input series becomes excessive. The recommended way to use **memse** is to begin by using **fft**, in order to estimate the model order. Typically this should be a small multiple of the number of peaks which you *believe* are present. Beware! **memse** will produce smooth spectral estimates for whatever model order you choose -- and they may be totally bogus if you choose incorrectly. Varying the model order can help to weed out some spurious features, but use extreme care when interpreting **memse** output given noisy input.

Options are:

-b *low high* [*low high ...*]
: Print power in the specified bands. Each *low* and *high* pair specifies the low and high frequency boundaries of the band of interest, in Hz. Multiple bands may be specified following a single **-b** option; only the last **-b** option has any effect. Also see **-s** below.

-f *frequency*
: Show the center frequency for each bin in the first column. The *frequency* argument specifies the input sampling frequency; the center frequencies are given in the same units.

-h
: Print a usage summary.

-n *n*
: Produce exactly *n* power estimates, evenly spaced in frequency from 0 up to half the input sampling frequency inclusive. The default depends on the length of the input series; it is designed to match *fft*'s defaults, to make it easy to compare outputs. You may wish to use values of *n* which are higher than the default in order to improve your estimates of the locations of sharp features in the spectrum; since this is not possible using **fft**, this feature is one of the main advantages of *memse*.

-o *n*
: Use an *n*th order model (i.e., up to *n* poles). Default: the square root of the number of input samples.

-P
: Generate a power spectrum (print squared magnitudes).

-s
: Print power in a standard set of frequency bands of interest for HRV analysis.

-w *window-type*
: Apply the specified window to the input data. *window-type* may be one of: 'Bartlett', 'Blackman', 'Blackman-Harris', 'Hamming', 'Hanning', 'Parzen', 'Square', and 'Welch'. The 'Square'

window type is equivalent to using no window at all; this is also variously known as a rectangular or Dirichlet window.

-z Add a constant to each input sample, chosen such that the mean value of the entire series is zero.

-Z Set the mean value of the inputs to zero as for **-z**, and detrend the series (set its mean first derivative to zero). This is equivalent to subtracting a best-fit (by least squares) line from the input data.

NOTES

Versions of **memse** released prior to September 1999 did not support the **-P** option, and did not normalize amplitudes with respect to the number of output points.

SEE ALSO

fft(1), **hrfft**(1), **lomb**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/psd/memse.c

NAME

mfilt – general-purpose median filter for WFDB records

SYNOPSIS

mfilt -l *length* [*options ...*]

DESCRIPTION

mfilt can be used to apply a median filter of any desired *length* to any desired section of a database record. The *length* is expressed in samples (i.e., each output sample is the median of *length* input samples). Median filters can be much more effective than any type of linear filter for removing impulse noise from signals; they are not particularly useful for removing persistent noise, however. Generally, the shortest effective median filter is the one that should be used, to minimize the aliasing effects resulting from the non-linear characteristics of the filter.

Options are:

-f *time* Filter from the specified *time* on the input record (default: start at the beginning of the record).

-h Print a usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-i *record*
Use the specified *record* for input (default: record 16).

-l *n* Use an *n*-point median.

-n *record*
Write the output signals to *record***.dat**, using the same specifications as the input signals, and create a header file for the specified *record*. This option overrides **-o** if both are used.

-o *record*
Use the specified *record* for output (default: record 16). If the output record header file specifies fewer signals than are present in the input, any extra input signals are discarded.

-t *time* Filter until the specified *time* on the input record (default: go to the end of the record).

In the present implementation, the same filter is applied to each input signal. For each output sample, an array of *length* input samples centered on the time of interest is sorted. (More efficient algorithms for finding the median exist, especially for large odd values of *length*; see, for example, *Numerical Recipes*.) If *length* is odd, the output is the middle value from the sorted array and there is no phase shift; otherwise, the output is the average of the two middle values from the array and there is a phase shift of one-half of the sampling interval. If necessary, the output is padded at the end to obtain equal numbers of input and output samples.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

Example

A 3-point median filter, applied to the first 5 minutes of record 100 to produce a new record 100m:

mfilt -l 3 -i 100 -n 100m

SEE ALSO

fir(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/mfilt.c

NAME

mrgann – merge annotation files

SYNOPSIS

mrgann -r *record* **-i** *ann1 ann2* **-o** *ann3* [*options ...*]

DESCRIPTION

mrgann reads a pair of annotation files (specified by *ann1*, *ann2*) for the specified *record* and writes a third annotation file (specified by *ann3*) for the same *record*. Typical applications of **mrgann** include combining annotation files that apply to different signals within a multi-signal record, and replacing a segment of an annotation file with annotations from another file (see the examples below). **mrgann** cannot concatenate annotation files from *different* records (e.g., segments of a multi-segment record); use **wfdbcollate**(1) for this purpose. If you wish to merge annotation files in order to be able to study or resolve the differences between them, **bxb**(1) (which can also merge annotation files using its **-o** or **-O** options) is almost certainly a better choice for such an application.

By default, the output annotation file contains copies of all annotations in each of the input files (if there are annotations with the same **time** and **chan** fields in each input file, however, only the annotation from *ann1* is copied). This behavior can be modified by command-line *options*, which include:

-c *n* Map (reset) the **chan** fields of all annotations from *ann1* to *n*. **chan** fields may contain integers between 0 and 255 inclusive; the **chan** field often specifies the signal number of the signal with which the annotation is associated. Specify **-c -1** to disable **chan** mapping for *ann1* (the default).

-C *n* Map (reset) the **chan** fields of all annotations from *ann2* to *n*. Specify **-C -1** to disable **chan** mapping for *ann2* (the default).

-h Print a usage summary.

-m0 *time*
Discard all annotations from both input annotators, beginning at *time*, until the time specified in the next **-m***x* option, or the end of the data if no other **-m***x* option is given.

-m1 *time*
Copy all annotations from *ann1*, and discard all annotations from *ann2*, beginning at *time*, until the time specified in the next **-m***x* option, or the end of the data if no other **-m***x* option is given.

-m2 *time*
Copy all annotations from *ann2*, and discard all annotations from *ann1*, beginning at *time*, until the time specified in the next **-m***x* option, or the end of the data if no other **-m***x* option is given.

-m3 *time*
Copy all annotations from *ann1* and *ann2*, beginning at *time*, until the time specified in the next **-m***x* option, or the end of the data if no other **-m***x* option is given. Annotations from *ann2* that match others from *ann1* in both the **time** and **chan** fields (after any *chan* mapping has been applied, see above) are discarded. This mode is the default.

-v Verbose mode (warn about simultaneous annotations with matching **chan** fields).

Note that options are interpreted in left-to-right order. For this reason, if you specify more than one **-m***x* option, as in the second example below, be sure to specify them in time order. It is also possible to use different **chan** mapping rules during different segments of the record; to do this, specify the appropriate **-c** or **-C** option(s) *before* the **-m***x* option that specifies the time when the new mapping rules are to be applied.

EXAMPLES

To merge three sets of annotations (named **a0**, **a2**, and **a3**, one for each of signals 0, 2, and 3 of record **999**), use the following commands:

mrgann -r 999 -a a0 a2 -o tmp -c 0 -C 2
mrgann -r 999 -a tmp a3 -o all -c -1 -C 3

Note that two passes are needed to merge three annotation files, since **mrgann** reads only two annotation files at a time. The first pass yields an intermediate result (annotator **tmp**); annotator **all** is the desired output. The **-c -1** option in the second command above disables **chan** mapping for annotations in **tmp**, which

have already been mapped as a result of the first command; this option could have been omitted, since **chan** mapping is disabled by default.

To replace any annotations in a set (named **old**) during the interval between 5 minutes and 6 minutes from the beginning of record **xyz**, with annotations from another set (named **new**), use the command:

mrgann -r xyz -a old new -o out -m1 0 -m2 5:0 -m1 6:0

In this command, the desired output is written to annotator **out** for record **xyz**. The **-m1 0** option overrides the default behavior and forces any **new** annotations that occur before the 5-minute mark to be discarded, while existing **old** annotations are copied to **out**. Beginning at the 5-minute mark, the **-m2 5:0** option changes the rules, and the **old** annotations are discarded as the **new** ones are copied. The rules are changed a third and final time at the 6-minute mark by the **-m1 6:0** option, which instructs *mrgann* to copy the remaining **old** annotations to **out**, while once again discarding any **new** annotations that occur during this interval.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

bxb(1), **setwfdb(1), wfdbcollate**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/mrgann.c

NAME

mxm – ANSI/AAMI-standard measurement-by-measurement annotation comparator

SYNOPSIS

mxm -r *record* **-a** *reference-annotator test-annotator* [*options ...*]

DESCRIPTION

This program implements the measurement-by-measurement comparison algorithm specified in ANSI/AAMI EC38:1998, the American National Standard for ambulatory ECGs, for evaluating heart rate measurements. Its use is not restricted to comparisons of these particular types of measurements, however; if other types of measurements (e.g., HRV measurements) are available, they may be compared in the same way by mxm.

Input to this program consists of two annotation files associated with the same *record*. One of these is designated the *reference* annotation file, the other the *test* annotation file.

Options include:

-f *time* Begin the comparison at the specified *time* (default: 5 minutes after the beginning of the record).

-h Print a usage summary.

-l *file* Append a line-format report to *file* (see below).

-L *file* Same as **-l** *file*.

-m *n* Compare measurement type *n* (default: *n* = 0).

-s *file* Append a standard-format report to *file* (see below).

-t *time* Stop the comparison at the specified *time* (default: the end of the record if it is defined, the end of the reference annotation file otherwise; if *time* is 0, the comparison ends when the end of either annotation file is reached).

-u Calculate unnormalized RMS measurement error (see below).

mxm reads the annotation files, ignoring all annotations except for those with *anntyp* = **MEASURE** and *subtyp* = *n* (where *n* is the measurement type selected using the **-m** option). The measurements to be compared are extracted from the *aux* fields of these annotations, which should contain strings with the measurements in **scanf**(3) **%lf** format (e.g., "85", "-12.4", "1.2e3"). A measurement error is calculated for each test measurement by comparing it with the reference measurement that is nearest in time. By default, **mxm** reports the normalized RMS measurement error (i.e., the square root of the sum of the squares of the differences between the test and reference measurements, divided by the sum of the reference measurements). If the **-u** option is given, **mxm** reports the unnormalized RMS measurement error (the square root of the sum of the squares of the differences between the test and reference measurements, divided by the number of test measurements); this may be useful if the measurement has a zero mean (or a mean value that is significantly smaller than the mean absolute value). The mean reference measurement that **mxm** reports is the mean of the reference measurements that are actually used in the comparison; since there is not necessarily a one-to-one correspondence between test and reference measurements, some reference measurements may not be included in the mean, and others may be included more than once.

If '**-**' is given as a *file* argument, reports are written on the standard output. If no options are specified, **mxm** writes standard reports on the standard output (equivalent to using the option **-s -**). The output generated by selecting **-l** or **-L** includes column headings only if a *file* other than '**-**' is specified, and only if the specified *file* does not already exist. In this way, **mxm** can be used repeatedly to build up a line-format table for multiple records, for further processing by **sumstats**(1).

DIAGNOSTICS

non-standard comparison selected

The **-f**, **-t**, and **-u** options modify the comparison in ways not permitted by the draft standard.

reference measurements have zero mean

Normalized RMS measurement error cannot be determined. Try using the **-u** option.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

bxb(1), **ecgeval**(1), **epicmp**(1), **rxr**(1), **setwfdb**(1), **sumstats**(1)

Evaluating ECG Analyzers (in the *WFDB Applications Guide*)

American National Standard ANSI/AAMI EC38:1998, Ambulatory Electrocardiographs; available from AAMI, 1110 N Glebe Road, Suite 220, Arlington, VA 22201 USA (http://www.aami.org/).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/mxm.c

NAME

nguess – guess the times of missing normal beats in an annotation file

SYNOPSIS

nguess -r *record* **-a** *input-annotator* [*options ...*]

DESCRIPTION

This program copies its input (a WFDB annotation file containing beat annotations), removing annotations of events other than sinus beats, and interpolating additional Q (unknown beat) annotations at times when sinus beats are expected. Intervals between sinus beats are predicted using a predictor array as described by Paul Schluter ("The design and evaluation of a bedside cardiac arrhythmia monitor"; Ph.D. thesis, MIT Dept. of Electrical Engineering, 1981). When the predictions are inconsistent with the known sinus beats, as may occur in extreme noise or in highly irregular rhythms such as atrial fibrillation, no interpolations are made.

Options for **nguess** include:

-f *time* Begin at the specified *time*. By default, **nguess** starts at the beginning of the record.

-h Print a usage summary.

-m *M* Insert Q annotations in the output at the inferred locations of sinus beats only when the input RR interval exceeds *M* times the predicted RR interval (default: *M* = 1.75). *M* must be greater than 1; its useful range is roughly 1.5 to 2.

-o *output-annotator*
Write output to the annotation file specified by *output-annotator* (default: **nguess**).

-t *time* Stop at the specified *time*.

It should be understood that, as the name of this program implies, the Q labels it generates represent, at best, good guesses about the times at which sinus beats may be expected. Ideally, one should avoid having to make such guesses, but some commonly-used techniques for study of heart rate variability (for example, conventional methods for power spectral density estimation in the frequency domain) require a uniformly sampled instantaneous heart rate signal, such as can be obtained using **tach**(1) to process the output of **nguess**. Other techniques, such as the Lomb periodogram method implemented by **lomb**(1), can obtain frequency spectra from time series with missing and irregularly spaced values, such as can be produced from a beat annotation file using **ihr**(1) without the need to use **nguess**. Use **nguess** only when necessary and do not expect it to perform miracles; as a rule of thumb, if the number of guesses (Q annotations) exceeds one or two percent of the number of known sinus beats (N annotations), be exceedingly wary of the guesses and consider using techniques such as **lomb**(1) that do not require the use of **nguess**. Also as a general rule, **nguess** works best when it is guessing the locations of sinus beats obscured by noise, or those of sinus beats that were inhibited by isolated premature ventricular beats; the underlying hypothesis of a quasi-continuous sinus rhythm, the basis not only of **nguess** but also of all other algorithms for reconstructing NN interval time series, is most suspect in the context of supraventricular ectopic beats (which may reset the SA node, thus interrupting the sinus rhythm) and consecutive ventricular ectopic beats.

The predictor array method is based on the observation that most of the short-term variability in normal sinus inter-beat (NN) intervals is due to respiratory sinus arrhythmia (RSA, the quasi-periodic modulation of heart rate by respiration, which is most notable in young, healthy subjects and decreases with age). Since respiration rate is (in humans and smaller mammals) substantially slower than heart rate, it is possible to estimate the length of the respiratory cycle in terms of some number of NN intervals. If, for example, heart rate is around 60 beats per minute and respiration rate is around 10 breaths per minute, one might expect that 6 NN intervals would correspond to one breath, and that the current interval might be particularly well-approximated by the sixth previous interval. Since we don't know the ratio between heart and respiration rate a priori, we can observe how well each of the previous **PBLEN** (a constant defined in **nguess.c**, see below) intervals predicts the current interval on average. Thus we have **PBLEN** predictors for each interval, some of which may be much better on average than others. At any time, we know which predictor is (locally) the best, and we can use that predictor to make a best guess of the location of the next sinus beat. In subjects with significant RSA, the best predictor may be determined by the length of the

respiratory cycle; in others, the previous beat may be a better predictor. For our purposes, it really doesn't matter which predictor is best, only that the mean error of the best predictor is small. If the next known sinus beat is at least 1.75 times as distant as the prediction, and if the predictions are reasonably good on average, 'nguess' asserts that a gap exists and fills it in with a Q annotation (or more than one, if the gap is sufficiently long).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

record.hea	header file
record.input-annotator	input annotation file (may contain any annotations)
record.output-annotator	output annotation file (contains N and Q annotations only)

SEE ALSO

ihr(1), **lomb**(1), **setwfdb**(1), **tach**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/nguess.c

NAME

nst – noise stress test for ECG analysis programs

SYNOPSIS

nst [*options* ...]

DESCRIPTION

nst adds calibrated amounts of noise from a *noise record* to ECGs (or other signals) from a *clean record*, generating an *output record* in WFDB format. Such output records make it possible to assess the noise tolerance of analysis programs.

Options include:

-a *annotator*
Use *annotator* as the reference annotator for the clean record. If the **-a** option is omitted, *atr* is used as the reference annotator. Reference annotations are used to determine the signal size as part of the noise level calibration, unless the **-p** option (see below) is used. Reference annotations are also copied to the output record.

-F *format*
Write output signals in the specified *format* (default: 16; for a list of valid formats, see **signal**(5)).

-h Print a usage summary.

-i *clean-record noise-record*
Read ECG (or other) signals from *clean-record*, and noise from *noise-record*. **nst** obtains these record names interactively if the **-i** option is omitted.

-o *output-record*
Create a record named *output-record* containing the input signals and added noise. **nst** obtains the name of the output record interactively if the **-o** option is omitted. If a header, signal, or reference annotation file for *output-record* already exists in the current directory, it will be overwritten.

-p *protocol*
Use *protocol* (the annotator name of an annotation file associated with the noise record) to define how noise is to be added to the signals (see below). If the **-p** option is omitted, **nst** generates a protocol annotation file.

-s *SNR* Set scale factors for noise such that the signal-to-noise ratio during noisy segments of the output record is *SNR* (in dB, see below). This option is ignored if a *protocol* is specified using **-p**.

Output signal generation

If the sampling frequencies of the clean and noise records differ by 10% or more, **nst** resamples the noise record (using **xform**(1)), producing a new noise record in the current directory. The name of the new record is that of the original (less any suffix beginning with an underscore), with a suffix consisting of an underscore followed by the sampling frequency of the new record. For example, if **nst** is asked to use AHA DB record *1001*, sampled at 250 Hz, and noise record *em*, sampled at 360 Hz, it first generates a new noise record named *em_250*, sampled at 250 Hz. If the noise record that **nst** would generate exists already, *nst* uses it without regenerating it. *nst* prints a warning if it is necessary to resample the noise record, or to substitute a previously resampled noise record.

Each ECG (or other) signal is paired and combined with a noise signal. A gain (*a*, a multiplicative scale factor) to be applied to the noise samples is set independently for each clean signal. If there are fewer noise signals than ECG signals, noise signals are paired with more than one clean signal as necessary. For example, if there are three clean signals and two noise signals, they are paired and combined as follows:

output signal 0 = clean signal 0 + *a*[0] * noise signal 0 + *b*[0]
output signal 1 = clean signal 1 + *a*[1] * noise signal 1 + *b*[1]
output signal 2 = clean signal 2 + *a*[2] * noise signal 0 + *b*[2]

The initial values of the gains, *a*, and offsets, *b*, are zero for all signals (i.e., no noise is added). In the protocol annotation file, the *time* field of each NOTE annotation specifies when gains are to be changed, and the *aux* field specifies new values for the gains (in **scanf**(3) **%lf** format, beginning with *a*[0]; values are

separated by white space within the *aux* field). The offsets, *b*, are recalculated at these times to cancel out step changes in signal levels when gains are changed. During the intervals between NOTE annotations in the protocol annotation file, gains and offsets are fixed.

If no protocol annotation file is specified, **nst** generates one using a standard protocol (a five-minute noise-free ''learning period'', followed by two-minute periods of noisy and noise-free signals alternately until the end of the clean record). The gains to be applied during the noisy periods are determined in this case by measuring the signal and noise amplitudes (see **Signal-to-noise ratios**, below).

Generation of the output signals ends at the time of the last NOTE annotation in the protocol annotation file, or at the end of the clean record, whichever comes first. If the noise record ends before that time, **nst** 'rewinds' the noise record to the beginning as necessary to obtain additional noise samples.

If a non-standard protocol is needed, it is probably easiest to run **nst** without the **-p** option to obtain a standard protocol annotation file. The standard file can be converted to text by **rdann**(1), edited as needed using any text editor, and converted back into annotation file format by **wrann**(1).

Signal-to-noise ratios

It is useful to characterize the noise level in a noise stress test in terms of the signal-to-noise ratio (SNR) during the noisy segments. SNR is commonly expressed in decibels (dB):

$$SNR = 10 \log (S/N)$$

where *S* is the power of the signal, and *N* is the power of the noise. If the **-p** option is omitted, **nst** measures *S* and *N*, and determines gains for the noise signals such that *SNR* matches the level specified using the **-s** option (or interactively).

The major difficulty in applying such a definition to the noise stress test is that most measurements of signal power are not particularly meaningful when applied to the ECG. A measurement based on mean squared amplitude, for example, will be proportional to the square of the heart rate. Such a measurement bears little relationship to a detector's ability to locate QRS complexes, which is typically related to the size of the QRS complex. A less significant problem is that unweighted measurements of noise power are likely to overestimate the importance of very low frequency noise, which is both common and (usually) not troublesome for detectors. In view of these issues, **nst** defines *S* as a function of the QRS amplitude, and *N* as a frequency-weighted noise power measurement. The definitions of *S* and *N* have been chosen such that SNRs given for noise stress tests will correspond roughly in terms of an intuitively defined 'signal quality' with SNRs such as those that may be encountered in other contexts.

To determine *S*, *nst* invokes **sigamp**(1) to read the reference annotation file for the ECG record and to measure the peak-to-peak amplitude of each of the first 300 normal QRS complexes (in each case, by measuring the range of amplitudes during a window from 50 ms before to 50 ms after the QRS annotation). The largest 5% and the smallest 5% of the measurements are discarded, and **sigamp** estimates the peak-to-peak QRS amplitude as the mean of the remaining 90% of the measurements. **nst** squares this peak-to-peak amplitude estimate and divides the result by 8 (correct for sinusoids, close enough for these purposes) to obtain the QRS ''power'' estimate, *S*.

To determine *N* for the unscaled noise signals, **sigamp** divides the first 300 seconds of the noise record into one-second chunks. For each chunk, **sigamp** determines the mean amplitude and the root mean squared difference, *n*, between the signal and this mean amplitude. As in the calculation of S, the largest 5% and the smallest 5% of the 300 measurements of *n* are discarded, and **sigamp** estimates the RMS noise amplitude as the mean of the remaining 90% of the measurements. *N* is the square of this estimate; if a noise signal is scaled by a gain, *a*, then *N* is scaled by the square of *a*. To obtain the desired *SNR* given *S* and *N*, **nst** solves for *a* in the equation:

$$SNR = 10 \log (S/(N * a^{**}2))$$

The calculations of *S*, *N*, and *a* are performed separately for each pair of clean and noise signals.

Noise records

Three noise records suitable for use with **nst** are available from http://www.physionet.org/physiobank/database/nstdb/ and are also provided in the *nstdb* directory of the MIT-BIH Arrhythmia Database CD-ROM. These contain noise of the types typically observed in ECG recordings. They were obtained using a Holter recorder and standard electrodes for ambulatory ECG monitoring, on an active subject. The electrodes

were placed on the limbs in locations chosen such that the subject's ECG is not visible in the recorded signals. Two signals were recorded simultaneously. Record *bw* contains primarily baseline wander, a low-frequency signal usually caused by motion of the subject or the leads. Record *em* contains electrode motion artifact (usually the result of intermittent mechanical forces acting on the electrodes), with significant amounts of baseline wander and muscle noise as well. Record *ma* contains primarily muscle noise (EMG), with a spectrum that overlaps that of the ECG, but which extends to higher frequencies. Electrode motion artifact is usually the most troublesome type of noise for ECG analyzers, since it can closely mimic characteristics of the ECG. For this reason, the remaining records in the *nstdb* directory consist of noise from record *em* mixed with clean ECGs by **nst**.

Although an early version of **nst** generated the records in the *nstdb/old* directory, the signal-to-noise ratios of these records were not determined using the definitions above. (Unfortunately, they were not calculated as stated in the *readme.doc* file on the first edition CD-ROM, either.) Using the definitions above, the signal-to-noise ratios (in dB) for the noisy portions of these records are as follows:

Record	*Signal 0*	*Signal 1*	*Record*	*Signal 0*	*Signal 1*
118_02	19.79	14.38	119_02	20.31	13.79
118_04	13.77	8.36	119_04	14.29	7.77
118_06	10.25	4.84	119_06	10.76	4.25
118_08	7.75	2.34	119_08	8.27	1.75
118_10	5.81	0.41	119_10	6.33	-0.19
118_12	4.23	-1.18	119_12	4.74	-1.77

Choosing 'clean' records

If the goal is to assess noise robustness, 'clean' records are best chosen from among those that can be analyzed without error (or with very low error rates). Given such a choice, any errors observed in analysis of **nst** output records can be attributed to the effects of the added noise, and not to any intrinsic properties of the signals.

Using nst output

The output records generated by **nst** may be analyzed in the same way as the clean records from which they were obtained. For ECG analyzers, programs such as **bxb**(1), **epicmp**(1), **mxm**(1), and **rxr**(1) may be useful for assessing the accuracy of analysis results. A series of **nst** output records with a range of signal-to-noise ratios may be used to determine how analyzer performance varies as a function of SNR. The parameter of greatest interest is usually the minimum value of SNR for which performance remains acceptable.

The standard protocol is designed to provide a fair yet difficult challenge to most analyzers. Segments of noise-free signals during the test period illustrate how rapidly the analyzer recovers its ability to analyze clean signals after having been presented with noisy signals.

Tests of multichannel analyzers should include records in which not all signals are equally noisy. Such records can be generated by **nst** with appropriately constructed protocol annotation files.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

bxb(1), **epicmp**(1), **mxm**(1), **rdann**(1), **rxr**(1), **setwfdb**(1), **sigamp**(1), **xform**(1), **wrann**(1), **signal**(5)

Moody, G.B., Muldrow, W.K., and Mark, R.G. A noise stress test for arrhythmia detectors. *Computers in Cardiology* **11**:381–384 (1984).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/nst.c

NAME

parsescp – parse SCP-ECG, optionally save in PhysioBank-compatible format

SYNOPSIS

parsescp [*options ...*]

DESCRIPTION

parsescp converts SCP-ECG output produced by SpaceLabs/Burdick ECG carts into more easily usable formats. It was written in 2000 and has been used to convert about a million ECGs collected at Boston's Beth Israel Deaconess Medical Center since then. **parsescp** can also be used to create deidentified SCP-ECG files, although it does not perform this function by default.

Options include:

-a Anonymize: copy standard input to standard output, removing protected health information (PHI). This option suppresses all other output.

-b Show baselines (residuals after template subtraction).

-f Force the input to be parsed even if it contains CRC errors.

-h Print a usage summary.

-l Low-pass filter (smooth) the output waveforms.

-o *record*
Set the record name (default: **ecg**) for output files.

-s *N* Shift templates by *N* samples before adding them to the baselines.

-S *N* Skip parsing of SCP-ECG section *N*.

-t Show templates; suppress baselines (complement of **-b** option).

-v Verbose mode: print a very detailed analysis of the SCP-ECG input, and write *record*.txt (specify *record* using **-o**).

-w Create a PhysioBank-compatible record (specify the record name using **-o**).

-x Show a hexadecimal data dump (implies **-v**).

-z Suppress final transients and zero-mean the ECGs.

Unless the **-a** option is used, this program produces at least these three files:

*record***.des**
(text) description (age, sex, recording bandwidth, measurements, diagnoses)

*record***.ecg** (binary) reconstructed ECGs (see comments in
parsescp.c, the **parsescp** source file, for format)

*record***.key**
(text) patient's name and ID (medical record number)

If invoked with the **-v** option, **parsescp** produces:

*record***.txt**
(text) reconstructed ECGs

With **-v**, **parsescp** also writes a (very) detailed analysis of the contents of the SCP-ECG input on the standard output.

If **parsescp** was compiled with the WFDB library, and if it is invoked with the **-w** option, it also produces a pair of PhysioBank-compatible output files:

*record***.dat**
(binary) signal file containing 12 continuous leads

*record***.hea**
(text) header file describing *record***.dat**

Supported SCP versions

This program was written using AAMI SCP-1999 (*Standard communications protocol for computer-assisted electrocardiography*, 25 October 1999 draft) as a reference for SCP format. It has been tested only with SCP records produced by SpaceLabs/Burdick ECG carts (these produce second-difference encoded data with reference beat subtraction using a single reference beat, Huffman encoded using the SCP standard Huffman table). Amplitude (unencoded) data and first-difference encoded data should be readable using this program, but these formats have not been tested. Use of custom Huffman tables is recognized but not otherwise supported. Use of multiple reference beats is recognized but not otherwise supported.

ECG signals in Spacelabs/Burdick SCP-ECG files

Spacelabs/Burdick ECG carts of the type for which this program was designed record 2 of the 3 Einthoven leads and all 6 precordial leads simultaneously for 10 seconds, at 500 samples per second per lead, with 16-bit precision over a range of +/-32.767 mV. Thus the sampling interval is 2 ms, and the amplitude resolution is 5 microvolts (5000 nanovolts) per ADC unit.

Note that although the SCP standard specifies how to record the sampling frequency and amplitude resolution in SCP-ECG files, the Spacelabs/Burdick carts don't do this, so **parsescp** assumes the sampling frequency and resolution above. **parsescp** will need modification in order to convert ECGs with other sampling frequencies or resolutions correctly.

ECG signals in parsescp's output files

This program derives the third Einthoven lead and the three augmented leads using the standard relationships among the leads:

III = II - I
aVR = -(I + II)/2
aVL = II/2 - III
aVF = I/2 + III

In all of its output formats, **parsescp** represents the samples of each signal as a sequence of unscaled integers, exactly as they appear in the original SCP-ECG input file. Thus, in the **.ecg**, **.txt**, and **.dat** output files, the unit of amplitude is equivalent to 5 microvolts (5000 nanovolts), as in the SCP input. If the recording is shorter than 10 seconds, or if a signal is missing and cannot be reconstructed from the relationships above, each missing sample is assigned a special value (WFDB_INVALID_SAMPLE, or -32768).

The **-l** and **-z** options modify the input values as noted above; if neither option is used, the output sample values are numerically identical to the input sample values.

The **.ecg** file contains selected and rearranged segments of the signals in the commonly-used layout of twelve 2.5 second segments arranged in groups of 4 above a continuous 10-second lead II. Each sample is represented as a big-endian 16-bit two's complement signed integer. The file begins with a 512-byte prolog containing the record name and recording date and time, which are HIPAA-defined protected health information (PHI) unless the input SCP-ECG has been deidentified. The prolog is followed by four "traces", each representing the same 10-second interval. The first three of these traces are made by concatenating 2.5 second segments (1250 samples) of each of the 12 leads, in this order:

(I aVR V1 V4)
(II aVL V2 V5)
(III aVF V3 V6)

The fourth trace is a continuous 10-second segment (5000 samples) of lead II.

The optional **.txt** and **.dat** files contain the ECG signals only (no metadata, and no PHI). The signals appear in the standard order:

I, II, III, aVR, aVF, aVL, V1, V2, V3, V4, V5, V6

In the **.txt** file, each line begins with a sample number (0 to 4999) and is followed by a sample from each of the 12 leads, in order. Each sample is represented as a base 10 numeral, with spaces inserted between samples so that the columns line up. Thus the sample numbers are in column 0, samples of lead I are in column 1, those of lead II are in column 2, etc.

In the **.dat** file, the first 24 bytes contain the first sample of each signal, in the standard order as for the **.txt** file. As in the **.ecg** file, each sample is represented as a big-endian 16-bit two's complement signed integer. The next 24 bytes contain the second sample of each signal, etc.

Other output files

The **.des** file contains a variety of information extracted from the SCP-ECG input file, in human-readable form. It does not contain the ECG signals themselves, or the patient's name or medical record number. Note that **.des** files made from SCP-ECG files that have not been anonymized will generally contain HIPAA-defined PHI (protected health information) such as the recording date and the patient's age (even if over 90).

The **.key** file contains the recording date and time, the patient's name, and the medical record number, if recorded in the input file.

The **.hea** file, if generated, contains metadata (information about the corresponding **.dat** file) only; it does not contain any PHI, even if the input was not anonymized. Age and sex are recorded in the **.hea** file if present in the input file, except that ages of 90 and more are recorded as 90. The recording date and time are not recorded in the **.hea** file.

Using parsescp to create deidentified SCP-ECG files

The SCP-ECG standard defines how to record a variety of information that includes elements defined by HIPAA as PHI (protected health information). These include the patient's name, medical record number, birth day and month, recording day and month, and (if the age is over 90) birth year and age.

If invoked with the **-a** option, **parsescp** reads the input SCP-ECG file and writes an anonymized (deidentified) version of it to the standard output. For example:

parsescp -a <12345678.scp >anonymous.scp

In this case, none of the other output files are produced.

parsescp removes all of the PHI as well as names of physicians and technicians, names of hospitals or clinics, and room numbers, replacing them with 'xxx'. It changes all dates to January 1, and if the age is over 90, it resets the age to 90 and the birth year to 90 years before the recording year. Finally, it recalculates the SCP-ECG CRCs so that the output is still a valid SCP-ECG file. Note that the original input file is not modified.

Note that **parsescp** does not deidentify other types of data (including its own **.des** and **.key** files); it can only deidentify SCP-ECG files.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

Examples

parsescp -o 12345 <12345.scp

The command above converts an SCP-ECG file named **12345.scp** into a set of three files (**12345.des**, **12345.ecg**, and **12345.key**), as described above. The argument following **-o** need not match the name of the input file as in this example, but such a choice may reduce opportunities for confusion.

parsescp -o 12345 -w <12345.scp

Same as the first example, but this command also creates a PhysioBank-compatible record named **12345** (consisting of two files named **12345.dat** and **12345.hea**).

parsescp -a <12345.scp >a001.scp

The final example reads its input (**12345.scp**), removes all PHI, and writes the deidentified data to a new SCP-ECG file (**a001.scp**).

Note that none of these commands modify the original input file (**12345.scp**).

SEE ALSO

rdsamp(1), **setwfdb**(1), **xform**(1), **signal**(5)

AUTHOR

George B. Moody (george@mit.edu) and Edna S. Moody

SOURCE

http://www.physionet.org/physiotools/wfdb/convert/parsescp.c

NAME

plot2d, plot3d – make 2-D or 3-D plots from text files of data, using **gnuplot**

SYNOPSIS

plot2d [*input-file*] [[*xcol*] *ycol*] [*options ...*]
plot3d [*input-file*] [[*xcol ycol*] *zcol*] [*options ...*]

DESCRIPTION

These UNIX shell scripts can be used to produce simple 2-D and 3-D plots using **gnuplot**(1) in batch (non-interactive) mode. **plot2d** was designed as a quick-and-dirty replacement for **plt**(1) (see http://www.physionet.org/physiotools/plt/). **plot2d** accepts a few of the most commonly-used **plt** options and produces similar plots. **plot3d** uses the same syntax as **plot2d**, but it produces simple 3-D plots (a capability not yet offered by **plt**).

The *input-file* should contain one or more space- or tab-separated columns of data per line, with each point on a line. Omit the *input-file* argument to read data from the standard input. (Note: since **gnuplot** cannot read data from a pipe, **plot2d** and **plot3d** save piped input in a temporary file before invoking **gnuplot**.)

xcol, *ycol*, and *zcol* specify the column numbers within the input file for the x, y, and z coordinates of the points to be plotted. The leftmost column is column 0 (this convention follows that used by **plt**, rather than that used by **gnuplot**). Omit the *xcol* argument to **plot2d** to use row numbers as abscissas; if *ycol* is also omitted, **plot2d** plots column 1 vs. column 0. When using **plot3d**, omit both *xcol* and *ycol* to generate x and y coordinates sequentially based on row numbers; a blank line in the input resets x and increments y in this case.

Options include:

-h Print help and exit (no plot is made).

-t *title* Use *title* as the title for the plot.

-x *label* Use *label* as the X-axis label.

-y *label* Use *label* as the Y-axis label.

-z *label* Use *label* as the Z-axis label (*plot3d* only).

-X *xmin xmax*
Plot x-coordinates between *xmin* and *xmax* only.

-Y *ymin ymax*
Plot y-coordinates between *ymin* and *ymax* only.

-Z *zmin zmax*
Plot z-coordinates between *zmin* and *zmax* only (*plot3d* only).

-T *printer*
Produce output on the specified PostScript *printer* (default: plot on-screen). Use **-T eps** to generate encapsulated PostScript on the standard output.

EXAMPLES

Create a text file with the following contents:

```
0  0  0
1  1  1
2  4  8
3  9 27
4 16 64
```

and call the file *powers*. Plot the first column vs. the second by:

plot2d powers 0 1 -t "Squares of small integers" -x "Integer" -y "Square"

The same file can be used to generate a number of different plots, by choosing different columns. To plot the third column vs. the first, try:

plot2d powers 2 0 -t "Marshmallows" -x "Mass (kg)" -y "Height (m)"

SEE ALSO

gnuplot(1), **plt**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/psd/plot2d
http://www.physionet.org/physiotools/wfdb/psd/plot3d
gnuplot: http://www.gnuplot.info/

NAME

plotstm – produce scatter plot of ST measurement errors on a PostScript device

SYNOPSIS

plotstm *file*

DESCRIPTION

plotstm reads a file of ST measurement errors produced by **epicmp**(1) using its **-S**, **-S0**, or **-S1** option, and generates a PostScript page description for a scatter plot of these data, as specified by ANSI/AAMI EC38 and ANSI/AAMI EC57. The standard output of **plotstm** may be printed directly on any PostScript device.

SEE ALSO

ecgeval(1), **epicmp**(1)

Evaluating ECG Analyzers

American National Standard ANSI/AAMI EC38:1998, Ambulatory Electrocardiographs

American National Standard ANSI/AAMI EC57:1998, Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms

The last two publications are available from AAMI, 1110 N Glebe Road, Suite 220, Arlington, VA 22201 USA (http://www.aami.org/).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/plotstm.c

NAME

plt – make 2-D plots

SYNOPSIS

plt [*data-spec*] [*data-file*] [[*xcol*] *ycol*] [*options ...*] [**-T lw | lwcat** [*lwcat-options*]]

DESCRIPTION

This man page is intended as a supplement to the command-line help provided by **plt** itself (using the **-h** option, see below). If you have not previously used **plt**, please look at the *plt Tutorial and Cookbook*, which is included in the **plt** package (see **SOURCES** below).

plt is a non-interactive (command line-driven) plotting utility. **plt** can produce publication-quality 2D plots in PostScript from easily-produced text or binary data files, and can also create screen plots under the X Window System.

All data presented to **plt** must be organized in rows and columns. Columns are numbered beginning with zero, and each column contains values for a variable that can be used as an abscissa (x coordinate), ordinate (y coordinate), or (with appropriate options described below) a grey level, color, or other plot attributes. Rows are numbered beginning with one, and each row contains a value for each column. Within a *data-file*, values are always arranged in row-major order (all elements of row 1, followed by all elements of row 2, etc.).

Usually, data must be in text form in order for **plt** to read them. Each non-empty, non-comment line (row) in the input should contain a value for each column that will be plotted; any additional values or other extra text at the end of a row will be ignored. Columns can be separated by any number of spaces or tabs. Commas and single or double quotation marks can also be used as column separators with current versions of **plt**, though not with older versions. It is not necessary to line up the values in each row. There may also be spaces or tabs at the beginning of a line, and these will also be ignored.

If no *data-file* is specified, **plt** reads data from its standard input. The command-line arguments *xcol* and *ycol* specify the column numbers for the abscissas and ordinates respectively. If only one column number is specified, it is taken as *ycol*, and **plt** generates a series of abscissas automatically. If the *data-file* contains no more than two columns, both *xcol* and *ycol* may be omitted.

By default, **plt** reads all rows of the *data-file* and scales the x and y axes so that all data can be plotted. An optional *data-spec*, a string beginning with a colon (:), can be used to select a subset of the rows in the *data-file*. For details on using a *data-spec*, and for information about reading binary data files using **plt**, see the *plt Tutorial and Cookbook*.

plt recognizes a large number of *options* for controlling and customizing plots. To see a summary of all options, run "**plt -h**"; if this command is followed by one or more strings (which should not begin with hyphens), **plt** prints one-line summaries of all options beginning with those strings only.

plt can read its options from command-line arguments, from a *format file* (specified using the **-f** option), or from a *format string* (supplied on the command line, following the **-F** option). When using format files or format strings, omit the hyphen (-) before each option.

Options

Following is a brief summary of **plt**'s options. Note that many options require arguments. **plt** chooses a suitable default for most such arguments if the argument is supplied as '**-**'. See the *plt Tutorial and Cookbook* for further details.

-p *plot-styles*

Specify style(s) for data plots. Available *plot-styles* include '**c**', '**C**', '**e+***c*', '**e-***c*', '**e:***c*', '**E+***n*', '**E-***n*', '**E:***n*', '**f**', '**i**', '**l**', '**m**', '**n**', '**N**', '**o**', '**O**', '**s***c*', '**S***n*', and '**t**'.

-s *elements*

Suppress *elements* of output. Elements that can be suppressed include '**e**' (erasing the screen or beginning a new page before plotting), '**a**' (anything associated with axes), '**x**' (anything associated with the x axis), '**y**' (anything associated with the y axis), '**g**' (the grid), '**m**' (x and y axis tick marks), '**n**' (x and y tick mark numbers), '**t**' (x and y axis labels and plot title), '**l**' (user-supplied labels), '**p**' (data plots), and '**f**' ("figures" -- boxes, line segments, arrows, and legends). In

addition, these *elements* modify the effects of any other elements that follow: '**X**' (restrict effects to x axis), '**Y**' (restrict effects to y axis), and '**A**' (apply effects to both axes); and the element '**C**' reenables all elements.

-X *xmin xmax*
Set the x-axis range (see also **-xa**).

-Y *ymin ymax*
Set the y-axis range (see also **-ya**).

-t *title* Set the title for the plot (enclose *title* in quotes if it contains whitespace or begins with '(' or '**[**').

-T *type* Specify the output *type*, which may be **xw** (X11 window, the default under Unix or Linux and not available under MS-Windows), or **lw** (PostScript, the default under MS-Windows).

-g *grid-mode*
Specify the grid style, which may be **in**, **out** (default), **both**, **none**, **sym** (make symmetric axes at top and right), **grid** (extend major ticks across the entire plot), **xgrid**, **ygrid**, or **sub** (extend all ticks across the entire plot).

-h [*option-prefix ...*]
Show help on options beginning with *option-prefix* (which should not begin with '**-**'). If *option-prefix* is omitted, show help on all options.

Within the next group of options, those with upper-case names ('**-A**', '**-B**', ...) use *window coordinates* between (0,0) and (1,1); those with lower-case names ('**-a**', '**-b**', ...) use *data coordinates*.

-a *x0 y0 x1 y1*
Draw an arrow to *(x0,y0)* from *(x1,y1)*.

-A *xw0 yw0 xw1 yw1*
Draw an arrow to *(xw0,yw0)* from *(xw1,yw1)*.

-b *x0 y0 x1 y1*
Draw a box with opposite corners at *(x0,y0) and (x1,y1)*.

-B *xw0 yw0 xw1 yw1*
Draw a box with opposite corners at *(xw0,yw0) and (xw1,yw1)*.

-c *x0 y0 x1 y1*
Connect points *(x0,y0)* and *(x1,y1)*.

-C *xw0 yw0 xw1 yw1*
Connect points *(xw0,yw0)* and *(xw1,yw1)*.

-d *x0 y0 x1 y1*
Draw a dark (filled) box with opposite corners at *(x0,y0) and (x1,y1)*.

-D *xw0 yw0 xw1 yw1*
Draw a dark (filled) box with opposite corners at *(xw0,yw0) and (xw1,yw1)*.

-l *x y tbc label-string*
Print *label-string* at *(x,y)*. The *tbc* argument is a two-character text box coordinate that specifies how the label is to be positioned relative to *(x,y)*; the default (**CC**) centers the string at *(x,y)*.

-L *xw yw tbc label-string*
As for **-l**, but using window coordinates *(xw,yw)*.

-w *configuration subwindow*
Confine the plot to a predefined window, specified by the arguments. *configuration* specifies the number of subwindows (panels), using one of '**m**' (1), '**b**' (2), or '**q**' (4), and *subwindow*' specifies which panel is to be plotted (0 or 1 for '**m**'; 0, 1, or 2 for '**b**'; or 0, 1, 2, 3, or 4 for '**q**'). In each case, subwindow 0 creates the frame of the entire plot, and the other subwindows refer to regions where data can be plotted. Use this option with '**-o**' or '**-s e**' to create multi-panel plots in stages without starting a new page or erasing the window before starting each new stage.

-W *xp0 yp0 xp1 yp1*
Define the region of the page in which to plot. The arguments are *page coordinates*; the page coordinates (0,0) and (1,1) correspond to the lower left and upper right corners of the page.

-f *format-file*
Read options from the specified *format-file*.

-fa *format-file*
Record the current axis parameters as options in the specified *format-file* (for use with a later **plt** command). The previous contents of *format-file*, if any, will be overwritten.

-F *format-string*
Read options from the specified *format-string*.

-o Suppress all output except data plots.

-cz *xfrom xincr*
Generate abscissas, beginning with *xfrom* (default: 0) and incrementing by *xincr* (default: 1) at each step.

-ex Don't exclude points outside axis limits.

-hl *x y tbc n file*
Print the next *n* (default: 1000) lines of the specified *file* as a label, placing the reference point for the first line of the label at data coordinates *(x,y)*. The *tbc* argument is defined as for **-l** and is applied to each line of the label. The *file* is opened when first used by **-hl** or **-vl**, and remains open, so that successive **-hl** or **-vl** options referring to the same *file* read and print successive lines. At most **MAXLABELFILES** (defined in **plt.h**, currently 6) *file*s of label strings can be open at once.

-vl *x y tbc n file*
As for **-hl**, but print the label in a vertical orientation (rotated 90 degrees counterclockwise).

-le *linenumber plotnumber* [*text*]
Define the specified *linenumber* in the legend (see also **-lp**). Line numbers in the legend begin with 0 (the top line); plot numbers also begin with 0 (these refer to the data plots, and are used here to determine the line style for the entry's sample plot segment). The *text* is printed to the right of the sample plot segment. To create an entry with more than one line of text, use additional **-le** options with different *linenumber*s as necessary, omitting the *plotnumber* (use '**-**') for all but the first. If the same data are plotted more than once in a single figure to create an overlay (for example, using symbols over line segments), an overlaid legend entry can be created using additional **-le** options with the same *linenumber* and different *plotnumbers*, omitting the *text* for all but the first.

-lp *xw0 yw0* [*boxscale* [*seglength* [*opaque*]]]
Define the window coordinates *(xw0, yw0)* of the upper left corner of the plot legend text, and other attributes for the plot legend (key). **plt** determines the size of the box it draws around the legend, but the calculated width of the box is multiplied by *boxscale*. The *seglength* option specifies the length of the sample plot segments, as a fraction of the x-axis length (default: 0.05). If *opaque* is '**yes**' (default), the background of the legend is opaque white; otherwise, the background is transparent (any previously drawn material remains visible through the legend box). Unless a **-lp** option is provided, no legend is printed.

-lx [*base* [*subticks*]]
Draw a logarithmic x-axis; *base* is the base of the logarithms (default: 10), and *subticks* is either '**yes**' or '**no**'. If the axis has a small number of major ticks, **plt** draws subticks by default; use the *subticks* argument to change **plt**'s default behavior.

-ly [*base* [*subticks*]]
Draw a logarithmic y-axis.

-tf *file* [*tbc*]
Load the text string array from the specified *file*. Each line of the *file* defines an element of the string array; using plot styles **c** or **t**, these strings can be plotted in the same manner as data points. The optional *tbc* specifies how the positions of the strings are to be modified when they are printed, in the same way as for **-l**; by default, the strings are centered on the coordinates specified for them.

-ts "*string0 string1 ...***"** [*tbc*]
Load the text string array from the quoted argument (whitespace separates strings in the array) rather than from a file; otherwise, this option is the same as **-tf**.

-fs "*string0 string1 ...***"**
Load the font string array from the quoted argument. Using appropriate plot style (**-p**) options, the strings can be used to change the font, line style (solid, dotted, dashed, etc.), or drawing color.

-x *string*
Set the x-axis title to *string* (which must be quoted if this option is used on the command line or if *string* begins with '(' or '[').

-xa *xmin xmax tick fmt tskip ycross*
Specify the x-axis range (as *xmin* to *xmax*); the interval between x-axis tick marks; the format, *fmt*, in which to print the numbers (e.g., "**%.3f**", "**%.2e**"; any format that **printf(3)** can use for printing floating-point numbers is acceptable); the number of ticks per labelled tick, *tskip*; and *ycross*, the point on the y-axis that the x-axis should cross, in y-units. Any of these parameters may be supplied as "-", which causes **plt** to choose a reasonable value based on the input data.

-xe *xmin-error xmax-error*
Use this option to specify the amount by which the x-axis range is allowed to exceed the range of x-values in the input data, when **plt** determines the x-axis range automatically.

-xm *tick-base*
Make x-axis ticks be multiples of the specified *tick-base*.

-xo *x-axis-offset*
Move the x-axis down by *x-axis-offset* (expressed as a fraction of the y-axis length).

-xr Draw the x-axis at the top of the plot

-xt *x label* [*tick-size*]
Add an extra labelled tick at the specified *x* position, and label it with the specified *label* (which may be any string). The optional *tick-size* argument specifies the length of the added tick, as a fraction of the default length for labelled ticks (e.g., a value of 1.5 makes the added tick 50 longer than the standard size).

-xts *x* [*tick-size*]
Force a labelled tick to appear on the x-axis at the specified *x* (the positions of the other labelled x-ticks are adjusted accordingly). *tick-size* is defined as for **-xt**.

-y *string*
Set the y-axis title to *string* (see **-x**).

-ya *ymin ymax tick fmt tskip xcross*
Set up the y-axis (see **-xa**).

-ye *ymin-error ymax-error*
Set the allowable error in the y-axis range (see **-xe**).

-ym *tick-base*
Make y-axis ticks be multiples of the specified *tick-base*.

-yo *y-axis-offset*
Move the y-axis to the left by *y-axis-offset* (expressed as a fraction of the x-axis length).

-yr Draw the y-axis at the right edge of the plot.

-yt *y label* [*tick-size*]
Add an extra labelled tick at the specified *y* position (see **-xt**).

-yts *y* [*tick-size*]
Force a labelled tick to appear on the y-axis at the specified *y* (see **-xts**).

-dev *pterm option*
Process *option* only if the value of **PTERM** is *pterm*. The **-dev** option may be useful in scripts that produce screen or printed plots in different formats.

-sf *name specification*
Create a new font group with the specified *name* and set its specifications (font, point size, color/grey level, line width, and line style). See the chapter titled *Colors, Line Styles, and Fonts* in the *plt Tutorial and Cookbook* for details.

-ch *height-factor width-factor*
Modify the height and width of all characters printed in the plot by the specified factors.

-size *fscl width height left-margin bottom-margin*
Specify the size and position of the plot on the page. The *width*, *height*, *left-margin*, and *bottom-margin* are specified in *inches* (1 inch = 25.4 mm). *fscl* is a factor applied to the point size of all printed characters, *independently* of the scaling applied to the rest of the plot. This option is effective for printed plots only.

Screen and printed plots

By default, **plt** makes an X11 screen plot. To make a printed plot, use the option **-T lw**, and pipe the output of **plt** to **lwcat**(1). Under Unix, GNU/Linux, or MacOS/X, **lwcat** uses the standard **lpr** print spooler to send **plt**'s output in PostScript format to the default printer. When running with a Cygwin/bash window under MS-Windows, or when using **lwcat**'s **-gv** option under Unix or Linux, the PostScript output is displayed on-screen using GhostScript (**GSView** under MS-Windows, or **gv** otherwise; these programs can save the output in a file or send it to a printer).

EXAMPLES

Create a text file with the following contents:

```
0  0  0
1  1  1
2  4  8
3  9 27
4 16 64
```

and call the file *powers*. Plot the first column vs. the second by:

plt powers 0 1 -t "Squares of small integers" -x "Integer" -y "Square"

The same file can be used to generate a number of different plots, by choosing different columns. To plot the third column vs. the first, try:

plt powers 2 0 -t "Marshmallows" -x "Mass (kg)" -y "Height (m)"

SEE ALSO

imageplt(1), **lwcat**(1), **pltf**(1)

The *plt Tutorial and Cookbook* (a book-length introduction to **plt**, included in the **plt** source package, and also available at **http://www.physionet.org/physiotools/plt/plt/doc/book.pdf**) contains many more examples.

AVAILABILITY

plt is available as part of PhysioToolkit (see **SOURCES** below) under the GPL.

AUTHORS

plt was originally written by Paul Albrecht, and is currently maintained by George B. Moody (**george@mit.edu**).

SOURCES

http://www.physionet.org/physiotools/plt/

NAME

pltf – make function plots

SYNOPSIS

pltf [*expression* [*xmin* [*xmax* [*xinc*]]]]

DESCRIPTION

pltf provides a simple way to use **bc**(1) and **plt**(1) to generate plots of many common functions of a single variable. The command-line arguments are interpreted according to their position; **pltf** asks for values for any missing arguments.

The first argument, *expression*, can be any expression valid as input to **bc**(1), with the additional feature that the variable **x** may appear anywhere in the expression where a number would be allowed by **bc**. Some examples of valid expressions are:

x^3+3*x^2+3*x+1

(x + 1)^3

s(sqrt(x^2))

The first two of these are equivalent; note that whitespace and parentheses are allowed in expressions, although it is necessary to enclose such expressions in double quotes (e.g., **"(x + 1)*e(x)"**) when entering them as command-line arguments in order to protect them from the shell. The last expression is the sine of the square root of x squared; see **bc**(1) for a complete list of available special functions, or invoke **pltf** with no command-line arguments to obtain a list.

The second and third arguments specify the domain of the function (the values over which **x** should vary), and the fourth argument specifies the **x**-increment (the difference between consecutive values of **x** for which the expression is to be evaluated).

pltf is a shell script that uses a helper application, **ftable**, to prepare input for **bc -l**. Invoke **ftable** directly (using the same arguments as for **pltf**) if you need to change the format of the plot or make a printed version of it. See the source for **pltf** to see how to do this.

SEE ALSO

imageplt(1), **plt**(1)

AVAILABILITY

pltf is available as part of the **plt** package in PhysioToolkit (see **SOURCES** below) under the GPL.

AUTHOR

George B. Moody (**george@mit.edu**)

SOURCES

http://www.physionet.org/physiotools/plt/plt/misc/pltf

http://www.physionet.org/physiotools/plt/plt/misc/ftable.c

NAME

pnnlist, pNNx – derive pNNx statistics from an annotation interval list or an annotation file

SYNOPSIS

pnnlist [*options ...*] **pNNx -r** *record* **-a** *input-annotator* [*options ...*]

DESCRIPTION

These programs derive *pNNx*, time domain measures of heart rate variability defined for any time interval *x* as the fraction of consecutive normal sinus (NN) intervals that differ by more than *x*. Conventionally, such measures have been applied to assess parasympathetic activity using *x* = 50 milliseconds (yielding the widely-cited *pNN50* statistic).

pnnlist

This program takes as standard input an annotation interval list, containing intervals in seconds and the (beat and non-beat) annotations that terminate each interval; and outputs on standard output each unique NN increment (x) in milliseconds, and the percentage of NN interval increments (pNNx) greater than x.

Options for **pnnlist** may include:

-h Print this usage summary.

-i *inc* Compute and output pNNx for x = 0, *inc*, *2*inc*, ... milliseconds.

-p Compute and output increments as percentage of initial intervals.

-s Compute and output separate distributions of positive and negative intervals.

pNNx

This shell script invokes **ann2rr**(1) and **pnnlist** to obtain pNNx statistics using a beat annotation file as input. The input file must be specified using the **-r** *record* and **-a** *annotator* arguments.

Options for **pNNx** may include any of those usable with **pnnlist**, as well as:

-f *time* Begin at the specified *time*. By default, **pNNx** starts at the beginning of the record.

-t *time* Stop at the specified *time*.

EXAMPLES

These commands are functionally identical:

```
ann2rr -r nsrdb/16265 -a atr -A -i s8 -w | pnnlist
pNNx -r nsrdb/16265 -a atr
```

Each of these commands reads the **atr** (reference) annotations for MIT-BIH Normal Sinus Rhythm Database (**nsrdb**) record 16265 (downloading them directly from PhysioNet if the annotation file has not previously been downloaded into a local **nsrdb** directory. These commands will then print each unique NN interval increment in milliseconds along with the percentage of intervals greater than that value. Both of the examples above produce the same output; the first few lines are shown below:

```
0       89.2738
7.8125  69.4564
15.625  53.3662
23.4375 40.8539
31.25   31.4265
39.0625 24.1817
46.875  18.4763
54.6875 14.1261
62.5    10.7312
70.3125 8.06025
78.125  6.09401
85.9375 4.56975
93.75   3.47841
101.562 2.66896
   .
   .
```

.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

ann2rr(1), **setwfdb**(1)

AUTHOR

Joe E. Mietus (joe at physionet dot org)

SOURCE

http://www.physionet.org/physiotools/pNNx/pNNx.src/pnnlist.c
http://www.physionet.org/physiotools/pNNx/pNNx.src/pNNx

NAME

pnwlogin – provide direct access to PhysioNetWorks for WFDB applications

SYNOPSIS

pnwlogin

DESCRIPTION

pnwlogin is a **bash** shell script that collects user credentials needed to access files in PhysioNetWorks projects, and provides them to WFDB applications. It also provides convenience functions, including validation of credentials and modification of the WFDB path (see **setdb**(1)) for convenient access to your private PhysioNetWorks projects as well as any active (shared) projects to which you belong.

After prompting for your PhysioNetWorks user name and password, **pnwlogin** starts a shell, making the login credentials available to any commands run within the shell. Applications that use WFDB library version 10.5.14 or later and libcurl 7.12.0 or later can make use of these credentials automatically with no modifications; you will not be prompted to enter them again while running within **pnwlogin**'s shell. Exit from the shell by typing a control-D or **exit** (as for any **bash** shell). Your credentials are never written to permanent storage, and **pnwlogin**'s in-memory copy of them is destroyed on exit.

First-time use:

You must be a member of an active project, or you must have created a private project, in order to use **pnwlogin** successfully. You cannot *create* a PhysioNetWorks account or join a PhysioNetWorks project using **pnwlogin**.

To get started, use your web browser to go to **https://physionet.org/users/**, click on "Create account", and follow the instructions. The process can be completed in a minute or two. Once you have an account, you will need to join an active project (follow the links on your PhysioNetWorks home page to visit the main pages of projects of interest for further information), or create a private project before you will be able to use **pnwlogin**.

ENVIRONMENT

PNWUSER

Your PhysioNetWorks user name (your email address).

PNWPASS

Your PhysioNetWorks password.

WFDB The database path: a list of locations (which may include names of local directories as well as URL prefixes) where WFDB-compatible input files may be found.

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://physionet.org/physiotools/wfdb/app/pnwlogin

NAME

pschart – produce annotated 'chart recordings' on a PostScript device

SYNOPSIS

pschart [[*options ...*] *script ...*]

DESCRIPTION

pschart produces high-quality annotated plots of WFDB records on PostScript devices. When rendered on a PostScript laser printer or phototypesetter, the plots closely resemble those that appear on pages 99–177 of the *MIT-BIH Arrhythmia Database Directory*.

pschart reads one or more *script* files containing newline-terminated commands. Its standard output is a PostScript file suitable for printing directly with no further processing. By default, **pschart** draws 'zero-width' lines; doing so typically reduces the printing time by a factor of three for a first-generation (300 dpi) laser printer while producing visually pleasing results. If the output is destined for a high-resolution (600 dpi or more) printer or phototypesetter, however, be sure to use the **-d** option (see below), or the traces and grid will be invisible (or nearly so).

Options:

-a *ann* Print annotations from annotator *ann* (default: 'atr'). To suppress annotation printing, use '**-a ""**'.

-A *ann* As for **-a**, but for a second annotator. The second set of annotations is shown below the first set.

-b *n* Set the binding offset to *n* millimeters (default: 0). The inside margin is increased by *n* mm, and the outside margin is decreased by the same amount.

-c *string*
Print 'Copyright © *string*' in the left page footer; *string* may include whitespace if it is quoted. The characters '*%*d', if included in *string*, are replaced by the current year. A default copyright notice is printed if no **-c** option is specified. To suppress printing the copyright notice, use '**-c ""**'.

-C Produce charts in color (default: black and white).

-Ca *r g b*
Draw annotations (if enabled) in the specified color. The color is specified using three numerical arguments (with values between 0 and 1 inclusive) that indicate the amounts of red, green, and blue respectively. Examples: **-Ca 0.5 0.5 1.0** produces light blue (the default obtained using **-C** only); **-Ca 0 0.5 0** produces a deep green color.

-Cg *r g b*
Draw the grid (if enabled) in the specified color. Default: red (1 0 0).

-Cl *r g b*
Draw labels and other non-annotation text in the specified color. Default: black (0 0 0).

-Cs *r g b*
Draw signals in the specified color. Default: deep blue (0 0 0.5).

-d *n* Set up for using a printer with a resolution of *n* dots per inch (default: *n* = 300, the typical resolution for laser printers). For a phototypesetter, *n* is typically 1200 or 2400. Note that *n* does not have to be correct in order to get properly scaled output; the value determines the granularity of the calculations made by **pschart** and the line width used by the printer, but not the scales.

-e Process even-numbered pages in a manner appropriate for two-sided printing. Even-numbered pages are printed with reversed page headers, and with the outside margin on the left (default: page headers are not reversed, and the inside margin is always on the left).

-E Generate EPSF format (encapsulated PostScript file format), suitable for inclusion in another PostScript file.

-g Print a 0.5 mV x 0.2 sec grid with 0.1 mV x 0.04 sec subticks under each strip (default: no grid). This grid is drawn using the **grid** procedure in the prolog file (see **ENVIRONMENT** below).

-G Print a 0.5 mV x 0.2 sec grid without subticks under each strip (default: no grid). This grid is drawn using the **Grid** procedure in the prolog file (see **ENVIRONMENT** below).

-h Print a usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-i *file* Print the (text) contents of *file* instead of the title in the title area of the first page of output. The text is printed in a monospaced font; use spaces rather than tabs in the text to align columns.

-l Label the signals in the margins next to each strip (default: no signal labels).

-L Print in landscape orientation (default: portrait orientation).

-m *inside outside top bottom*
Specify page margins in millimeters. Defaults: *top* and *bottom*, 25 mm; *inside* and *outside*, 25−37.5 mm (half of the difference between the page width and the default strip width). The default strip width is the largest multiple of 25 mm that is at least 50 mm less than the page width. Note that page headers and footers, time stamps, and signal labels are printed in the margins. Also note that hardware-enforced, printer-specific margins are not included; the margins specified using **-m** apply to the imageable area, and not necessarily to the physical page.

-M Print marker bars across the signals to show the locations of beat annotations (equivalent to **-M1**).

-M*barstyle*
Set marker bar and annotation format (note: no space between **-M** and *barstyle*). Legal values for *barstyle*: 0 (no bars); 1 (bars across all signals); 2 (bars across attached signal, annotations at center); 3 (bars across attached signal, annotations above bars). Default: *barstyle* = 0.

-n *n* Use *n* as the number of the first page (default: 1). Use '**-n 0**' (or any negative value for *n*) to suppress page numbering.

-p Pack sufficiently short strips side-by-side (default: print each strip centered between the inside and outside margins in a row by itself).

-P *pagesize*
Specify the size of the output pages to be printed. Legal values for *pagesize* are: 'letter' (8.5" x 11", 216 mm x 279 mm; imageable area 209 mm x 272 mm), 'lwletter' (8.5" x 11", 216 mm x 279 mm; imageable area 203 mm x 277 mm), 'legal' (8.5" x 14", 216 mm x 356 mm; imageable area 209 mm x 348 mm), 'legal13' (8.5" x 13", 216 x 330 mm; imageable area 209 mm x 322 mm), 'A4' (8.27" x 11.69", 210 mm x 297 mm; imageable area 202 mm x 289 mm), 'A5' (5.84" x 8.27", 148 mm x 210 mm; imageable area 140 mm x 202 mm); 'B4' (9.84" x 13.9", 250 mm x 353 mm; imageable area 249 mm x 356 mm), 'B5' (6.93" x 9.84", 176 mm x 250 mm; imageable area 173 mm x 249 mm), or '*width*x*height*' (where *width* and *height* are the width and height of the imageable area in millimeters). 'lwletter' is the standard letter size for the Apple LaserWriter; all of the other predefined page sizes are those used by the Sun SPARCprinter. Note that some printers may require non-standard PostScript code to select non-standard page sizes; in such cases, it may be necessary to customize the prolog file (see **FILES**). Default: letter size.

-r Print ''Record *xxx*'' as the first part of the title of each strip, where *xxx* is the record name.

-R Print a record name as part of the header on each page. If strips from two or more records are printed on one page, the name of the last record is printed.

-s *signal-list*
Print only the signals named in the *signal-list* (one or more signal numbers or names, separated by spaces; default: print all signals).

-S *scale-mode timestamp-mode*
Print scales and timestamps in the specified modes. Legal values for *scale-mode*: 0 (no scales); 1 (mm/unit in footers); 2 (units/tick in footers); 3 (mm/unit above strips); 4 (units/tick above strips); 5 (mm/unit within strips); 6 (units/tick within strips). Legal values for *timestamp-mode*: 0 (no timestamps); 1 (elapsed times only); 2 (absolute times if defined, elapsed times otherwise).

Defaults: *scale-mode* = 1, *timestamp-mode* = 2.

-t *n* Set the time scale to *n* millimeters per second (default: *n* = 12.5, half of the standard scale for chart recorders).

-T *title* Set the page title to *title* (which may include whitespace if quoted). If no **-T** option is specified, the page title is constructed from the date of the last recording on the page, if defined, or today's date otherwise. To suppress printing the page title, use '**-T ""**'.

-u Generate 'unstructured' PostScript as a workaround for a bug in the Adobe TranScript software (also see **ENVIRONMENT** below). Default: generate structured PostScript, suitable for processing by page-selection or page-reversal post-processors.

-v *n* Set the voltage (ordinate) scale to *n* millimeters per millivolt. Signals that do not have units of millivolts (as specified in the record's header file) are scaled proportionately, as specified by the calibration file (see **wfdbcal**(5)). The default scale is 5 mm/mV, half of the standard scale for chart recorders.

-V Verbose mode (echo each command as it is read from the script file).

-w *n* Set the line width for signals, grid lines, and marker bars to *n* mm. Default: 0 (the narrowest possible width; note that some devices may not render zero-width lines correctly).

-1 Print only the first character of each comment annotation.

Color output

If none of the **-C** options is used, output is in black and white. If any color option is used, output is in the default colors (light blue annotations, red grid, black labels, deep blue signals) unless overridden by one or more of the **-Ca**, **-Cg**, **-Cl**, or **-Cs** options. Color output can be rendered in greyscale by monochrome PostScript printers, although black-and-white output may look better in such cases.

Scripts:

Any argument that is not an option or an option argument is taken as the name of a script of newline-terminated commands to be executed by **pschart**. If the script name is '-', **pschart** reads commands from the standard input. Options that follow a script name are not applied to the processing of that script, so it is possible to use two or more scripts with different sets of options in a single run. Standard commands are of the following form:

record time title

in which *record* is the name of the record for which a strip is to be printed, *time* indicates the time of the left edge of the strip to be printed, and *title* is a description to be printed above the strip. Fields are separated by spaces or tabs. If the *time* field contains a hyphen ('-'), the portion that precedes the hyphen is taken as the time of the left edge of the strip, and the portion that follows the hyphen indicates the end of the desired segment; additional strips continuous with the first are printed if necessary. Unless the **-p** option is specified, strips that are less than the full width of the page are centered within the margins. The *title* field may include embedded spaces or tabs, or it may be omitted. A totally empty command line specifies a page break, i.e., it causes **pschart** to put the next strip at the top of a new page, even if the current page is not full.

ENVIRONMENT

The environment variable **PSCHARTPRO** can be used to name an alternate prolog file (see below) for custom formats. The environment variable **TRANSCRIPTBUG** may be set (to any value) to generate 'unstructured' PostScript by default (see the **-u** option above). It may be necessary to set and export the shell variables **WFDB** and **WFDBCAL** (see **setwfdb**(1)).

FILES

/usr/local/lib/ps/pschart.pro

default PostScript prolog file.

/usr/local/lib/ps/12lead.pro

alternative PostScript prolog file, suitable for printing standard 12-lead diagnostic ECGs (10 seconds, 4 traces, with the top three traces divided into 2.5 second segments by marker bars). This file redefines the grid drawn by the **-G** option (see the **Grid** procedure for details).

BUGS

On older PostScript printers, output may be quite slow. A full page, with grids and default scales, typically takes about 3 minutes to render on an Apple LaserWriter, or about 6 minutes on a Linotronic 1200 dpi phototypesetter. Most modern printers can render **pschart** output at nearly full speed.

If the record you wish to plot is sampled at a very high rate relative to the printer resolution (i.e., if one sample interval would appear on the page as much less than the distance between pixels), you may wish to use **xform**(1) to decimate to a lower frequency for efficiency's sake. In extreme cases, this may be necessary to avoid running out of memory in your PostScript printer.

Specifying EPSF output using the **-E** option does not prevent **pschart** from producing multi-page output, which is not permitted in EPSF. You should make sure that your output fits entirely onto one page (most easily verified using the **-V** option) before including it in another document. Note that the bounding box calculated by **pschart** covers the entire width of the page and most of its height (excluding only about half of the top and bottom margins, so that the header and footer material is included), even if only a small portion of the page contains plots. If you wish to fit such a plot into another document with a minimum of empty space around it, you may either edit the bounding box comment in the **pschart** output, or specify a page size that closely matches the size of your plot. The document in which **pschart** output is included can arbitrarily rescale the plot, so that scales expressed in mm/unit cannot be relied upon.

Under MS-DOS, a bug in **command.com** makes it impossible to pass an empty string in the argument list of a command, so that **-a ""**, **-c ""**, and **-T ""** do not work as described above. Type a space between the quotation marks to avoid this bug, or use one of the UNIX shells that have been ported to MS-DOS instead of **command.com**.

There are too many options. Invoke **pschart** with no arguments for a brief summary of options.

AVAILABILITY

This program is provided in the *app* directory of the WFDB Software Package. Run **make** in that directory to compile and install it if it have not been installed already.

The PhysioNet ATM (http://physionet.org/cgi-bin/ATM) provides web access to **pschart** (select **Plot waveforms** from the Toolbox).

SEE ALSO

psfd(1), **setwfdb**(1), **wave**(1), **xform**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/app/pschart.c
http://www.physionet.org/physiotools/wfdb/app/pschart.pro

NAME

psfd – produce annotated 'full-disclosure' plots on a PostScript device

SYNOPSIS

psfd [[*options ...*] *script ...*]

DESCRIPTION

psfd produces high-quality annotated 'full-disclosure' plots of WFDB records on PostScript devices. When rendered on a PostScript laser printer or phototypesetter, the plots closely resemble those that appear on pages 2–97 of the *MIT-BIH Arrhythmia Database Directory*.

psfd reads one or more *script* files containing newline-terminated commands. Its standard output is a PostScript file suitable for printing directly with no further processing. By default, **psfd** draws 'zero-width' lines; doing so typically reduces the printing time by a factor of three for a first-generation (300 dpi) laser printer while producing visually pleasing results. If the output is destined for a high-resolution (600 dpi or more) printer or phototypesetter, however, be sure to use the **-d** option (see below), or the traces and grid will be invisible (or nearly so).

Options:

-a *ann* Print annotations from annotator *ann* (default: 'atr'). To suppress annotation printing, use '**-a ""**'.

-A *ann* As for **-a**, but for a second annotator. The second set of annotations is shown below the first set.

-b *n* Set the binding offset to *n* millimeters (default: 0). The inside margin is increased by *n* mm, and the outside margin is decreased by the same amount.

-c *string*
Print 'Copyright © *string*' in the left page footer; *string* may include whitespace if it is quoted. The characters '%d', if included in *string*, are replaced by the current year. A default copyright notice is printed if no **-c** option is specified. To suppress printing the copyright notice, use '**-c ""**'.

-C Produce charts in color (default: black and white).

-Ca *r g b*
Draw annotations (if enabled) in the specified color. The color is specified using three numerical arguments (with values between 0 and 1 inclusive) that indicate the amounts of red, green, and blue respectively. Examples: **-Ca 0.5 0.5 1.0** produces light blue (the default obtained using **-C** only); **-Ca 0 0.5 0** produces a deep green color.

-Cg *r g b*
Draw the grid (if enabled) in the specified color. Default: red (1 0 0).

-Cl *r g b*
Draw labels and other non-annotation text in the specified color. Default: black (0 0 0).

-Cs *r g b*
Draw signals in the specified color. Default: deep blue (0 0 0.5).

-d *n* Set up for using a printer with a resolution of *n* dots per inch (default: *n* = 300, the typical resolution for laser printers). For a phototypesetter, *n* is typically 1200 or 2400. Note that *n* does not have to be correct in order to get properly scaled output; the value determines the granularity of the calculations made by **psfd** and the line width used by the printer, but not the scales.

-e Process even-numbered pages in a manner appropriate for two-sided printing. Even-numbered pages are printed with reversed page headers, and with the outside margin on the left (default: page headers are not reversed, and the inside margin is always on the left).

-E Generate EPSF format (encapsulated PostScript file format), suitable for inclusion in another PostScript file.

-g Print a grid with 1-second tick marks at the top of each page and below the last strip on each page (default: no grid).

-h Print a usage summary.

-H *n* Allot approximately *n* millimeters of vertical space on the page for each trace (default: *n* = 7.5).

-l Label the signals in the margins next to each strip (default: no signal labels).

-L Print in landscape orientation (default: portrait orientation).

-m *inside outside top bottom*
Specify page margins in millimeters. Defaults: *top* and *bottom*, 25 mm; *inside* and *outside*, 25–37.5 mm (half of the difference between the page width and the default strip width). The default strip width is the largest multiple of 25 mm that is at least 50 mm less than the page width. Note that page headers and footers, time stamps, and signal labels are printed in the margins. Also note that hardware-enforced, printer-specific margins are not included; the margins specified using **-m** apply to the imageable area, and not necessarily to the physical page.

-M Print marker bars across the signals to show the locations of beat annotations (equivalent to **-M1**).

-M*barstyle*
Set marker bar and annotation format (note: no space between **-M** and *barstyle*). Legal values for *barstyle*: 0 (no bars); 1 (bars across all signals); 2 (bars across attached signal, annotations at center); 3 (bars across attached signal, annotations above bars). Default: *barstyle* = 0.

-n *n* Use *n* as the number of the first page (default: 1). Use '**-n 0**' (or any negative value for *n*) to suppress page numbering.

-N Print counter values after time stamps in the left margin.

-P *pagesize*
Specify the size of the output pages to be printed. Legal values for *pagesize* are: 'letter' (8.5" x 11", 216 mm x 279 mm; imageable area 209 mm x 272 mm), 'lwletter' (8.5" x 11", 216 mm x 279 mm; imageable area 203 mm x 277 mm), 'legal' (8.5" x 14", 216 mm x 356 mm; imageable area 209 mm x 348 mm), 'legal13' (8.5" x 13", 216 x 330 mm; imageable area 209 mm x 322 mm), 'A4' (8.27" x 11.69", 210 mm x 297 mm; imageable area 202 mm x 289 mm), 'A5' (5.84" x 8.27", 148 mm x 210 mm; imageable area 140 mm x 202 mm); 'B4' (9.84" x 13.9", 250 mm x 353 mm; imageable area 249 mm x 356 mm), 'B5' (6.93" x 9.84", 176 mm x 250 mm; imageable area 173 mm x 249 mm), or '*width*x*height*' (where *width* and *height* are the width and height of the imageable area in millimeters). 'lwletter' is the standard letter size for the Apple LaserWriter; all of the other predefined page sizes are those used by the Sun SPARCprinter. Note that some printers may require non-standard PostScript code to select non-standard page sizes; in such cases, it may be necessary to customize the prolog file (see **FILES**). Default: letter size.

-r Print a record name as part of the header on each page. If strips from two or more records are printed on one page, the name of the last record is printed.

-R Same as **-r**.

-s *signal-list*
Print only the signals named in the *signal-list* (one or more signal numbers or names, separated by spaces; default: print all signals).

-S *scale-mode timestamp-mode*
Print scales and timestamps in the specified modes. Legal values for *scale-mode*: 0 (no scales); 1 (mm/unit in footers); 2 (units/tick in footers). Legal values for *timestamp-mode*: 0 (no timestamps); 1 (elapsed times only); 2 (absolute times if defined, elapsed times otherwise). Defaults: *scale-mode* = 1, *timestamp-mode* = 2.

-t *n* Set the time scale to *n* millimeters per second (default: *n* = 2.5, one-tenth of the standard scale for chart recorders).

-T *title* Set the page title to *title* (which may include whitespace if quoted). If no **-T** option is specified, the page title is constructed from the date of the last recording on the page, if defined, or today's date otherwise. To suppress printing the page title, use '**-T ""**'.

-u Generate 'unstructured' PostScript as a workaround for a bug in the Adobe TranScript software (also see **ENVIRONMENT** below). Default: generate structured PostScript, suitable for processing by page-selection or page-reversal post-processors.

-v *n* Set the voltage (ordinate) scale to *n* millimeters per millivolt. Signals that do not have units of millivolts (as specified in the record's header file) are scaled proportionately, as specified by the calibration file (see **wfdbcal**(5)). The default scale is 1 mm/mV, one-tenth of the standard scale for chart recorders.

-V Verbose mode (echo each command as it is read from the script file).

-w *n* Set the line width for signals, grid lines, and marker bars to *n* mm. Default: 0 (the narrowest possible width; note that some devices may not render zero-width lines correctly).

-x Extend the last strip of each record up to 10% if necessary to avoid printing a short strip at the end. (This option may be used to obtain plots like those in the *MIT-BIH Arrhythmia Database Directory*.)

-1 Print only the first character of each comment annotation.

Scripts:

Any argument that is not an option or an option argument is taken as the name of a script of newline-terminated commands to be executed by **psfd**. If the script name is '-', **psfd** reads commands from the standard input. Options that follow a script name are not applied to the processing of that script, so it is possible to use two or more scripts with different sets of options in a single run. Standard commands are of the following form:

record time

in which *record* is the name of the record for which a 'full disclosure' plot is to be printed, and *time* indicates the starting time (and, optionally, the stop time) of the plot. Anything that follows the *time* field in a command is ignored. Fields are separated by spaces or tabs. If the *time* field contains a hyphen ('-'), the portion that precedes the hyphen is taken as the starting time of the plot, and the portion that follows the hyphen indicates the stop time. A totally empty command line causes **psfd** to put the next plot at the top of a new page, even if the current page is not full. **pschart**(1) command scripts are usable by **psfd**; note, however, that the programs use different conventions for interpreting a missing stop time, and that strip titles are not printed by **psfd**.

ENVIRONMENT

The environment variable **PSFDPRO** can be used to name an alternate prolog file (see below) for custom formats. The environment variable **TRANSCRIPTBUG** may be set (to any value) to generate 'unstructured' PostScript by default (see the **-u** option above). It may be necessary to set and export the shell variables **WFDB** and **WFDBCAL** (see **setwfdb**(1)).

FILES

/usr/local/lib/ps/psfd.pro

default PostScript prolog file.

BUGS

On older PostScript printers, output may be quite slow. A full page, with grids and default scales, typically takes about 3 minutes to render on an Apple LaserWriter, or about 6 minutes on a Linotronic 1200 dpi phototypesetter. Most modern printers can render **psfd** output at nearly full speed.

For a 300 dpi printer, a typical full page of output will be about 80 Kbytes. Expect this to increase approximately linearly with the printer resolution.

The signals are decimated to obtain samples that are spaced by intervals approximating one pixel. To obtain this result, the signals are first digitally low-pass filtered by **psfd**; in general, this has no significant effect on the appearance of the plots other than a slight improvement in legibility for signals contaminated by high-frequency noise. To get an idea of the high-frequency content of the signals, use **pschart**(1).

Specifying EPSF output using the **-E** option does not prevent **psfd** from producing multi-page output, which is not permitted in EPSF. You should make sure that your output fits entirely onto one page (most easily verified using the **-V** option) before including it in another document. Note that the bounding box

calculated by **psfd** covers the entire width of the page and most of its height (excluding only about half of the top and bottom margins, so that the header and footer material is included), even if only a small portion of the page contains plots. If you wish to fit such a plot into another document with a minimum of empty space around it, you may either edit the bounding box comment in the **psfd** output, or specify a page size that closely matches the size of your plot. The document in which **psfd** output is included can arbitrarily rescale the plot, so that scales expressed in mm/unit cannot be relied upon.

Under MS-DOS, a bug in **command.com** makes it impossible to pass an empty string in the argument list of a command, so that **-a ""**, **-c ""**, and **-T ""** do not work as described above. Type a space between the quotation marks to avoid this bug, or use one of the UNIX shells that have been ported to MS-DOS instead of **command.com**.

There are too many options. Invoke **psfd** with no arguments for a brief summary of options.

SEE ALSO

pschart(1), **setwfdb**(1), **wave**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/app/psfd.c
http://www.physionet.org/physiotools/wfdb/app/psfd.pro

NAME

rdann – read a WFDB annotation file

SYNOPSIS

rdann -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

rdann reads the annotation file specified by *record* and *annotator*, and writes a text-format translation of it on the standard output, one annotation per line. The output contains (from left to right) the time of the annotation in hours, minutes, seconds, and milliseconds; the time of the annotation in samples; a mnemonic for the annotation type; the annotation **subtyp**, **chan**, and **num** fields; and the auxiliary information string, if any (assumed to be a null-terminated ASCII string).

Options include:

-c *chan* Print only those annotations with **chan** fields that match *chan*.

-e Print annotation times as elapsed times from the beginning of the record (default: **rdann** prints absolute times if the absolute time of the beginning of the record is defined, and elapsed times otherwise, unless the **-x** option has been given).

-f *time* Begin at the specified *time*. By default, **rdann** starts at the beginning of the record; if modification labels are present, they are not printed unless **-f 0** is given explicitly, however.

-h Print a usage summary.

-n *num* Print only those annotations with **num** fields that match *num*.

-p *type* [*type ...*]
Print annotations of the specified *types* only. The *type* arguments should be annotation mnemonics (e.g., **N**) as normally printed by **rdann** in the third column. More than one **-p** option may be used in a single command, and each **-p** option may have more than one *type* argument following it. If *type* begins with "-", however, it must immediately follow **-p** (standard annotation mnemonics do not begin with "-", but modification labels in an annotation file may define such mnemonics).

-s *sub* Print only those annotations with **subtyp** fields that match *sub*.

-t *time* Stop at the specified *time*.

-v Print column headings.

-x Use an alternate time format for output (the first three columns are the elapsed times in seconds, in minutes, and in hours, replacing the *hh:mm:ss* and sample number columns in the default output). Note that this format is incompatible with **wrann**.

The **-f** and **-t** options may be used to select a portion of an annotation file for printing. Their arguments are usually given in standard *time* (*hh:mm:ss*) format; see the description of *strtim* in the *WFDB Programmer's Guide*, as well as the comments below, for other formats.

Annotation numbers beginning with 0 are implicitly assigned by **rdann** to each annotation in an annotation file, and beat numbers beginning with 0 are assigned to each QRS annotation. If the argument of the **-f** option begins with 'a', it is taken to be the annotation number of the first annotation to be printed; if it begins with 'b', it is taken to be the beat number of the first beat annotation to be printed (any non-QRS annotations that immediately precede this annotation are also printed). Arguments of the **-t** option beginning with 'a' or 'b' similarly specify the annotation number or beat number following the last to be printed. If the argument of the **-t** option begins with '#', it is taken as the number of QRS annotations to be processed; note that not all of those processed will necessarily be printed, if the **-p** option is used to select only a subset of annotation types to be printed.

Note that the **-e** and **-x** options are mutually exclusive; if both are given, only the last one is effective.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

Example

rdann -a atr -r 200 -f 0 -t 5:0 -p V

This command prints on the standard output all **V** (premature ventricular contraction) annotations in the first five minutes of the *atr* (reference annotation) file for record 200.

CD-ROM Versions

The first edition of the MIT-BIH Arrhythmia Database CD-ROM, the first and second editions of the European ST-T Database CD-ROM, and the first edition of the MIT-BIH Polysomnographic Database CD-ROM contain versions of **rdann** that use an older command syntax (still supported by the current version but not described here). Refer to *bin.doc* in the CD-ROM directory that contains **rdann** for further information.

AVAILABILITY

This program is provided in the *app* directory of the WFDB Software Package. Run **make** in that directory to compile and install it if it have not been installed already.

The PhysioNet ATM (http://physionet.org/cgi-bin/ATM) provides web access to **rdsamp** (select **Show annotations as text** from the Toolbox).

SEE ALSO

rdsamp(1), **setwfdb**(1), **wrann**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/rdann.c

NAME

rdedfann – extract annotations from an EDF+ file

SYNOPSIS

rdedfann -r *edffile* [*options ...*]

DESCRIPTION

This program prints the annotations from an EDF+ file in the same format as **rdann** does for WFDB-compatible annotation files.

Options include:

-F *frequency*
: Set the sampling frequency to *frequency* (in Hz).

-h Print a brief usage summary.

-v Verbose mode (print column headings).

-x Print EDF+ annotation text in 'aux' rather than 'anntyp' column.

Note that the annotation mnemonics in EDF+ files do not in general match those used in WFDB-compatible annotation files, so that it will often be desirable to translate those that come from EDF+ files before converting the text with **wrann**. For example, this command can be used to extract annotations from **foo.edf**, change the EDF+ annotation type "QRS" to the WFDB type "N", and then produce a WFDB-compatible annotation file **foo.edf.qrs**:

rdedfann -r foo.edf | sed s/QRS/N/ | wrann -r foo.edf -a qrs

Recent versions of **wrann** copy unrecognized mnemonics into the **aux** field, setting the annotation type to **NOTE**, so it is no longer essential to translate mnemonics as described above before processing **rdedfann**'s output with **wrann**.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

AVAILABILITY

This program is provided in the *convert* directory of the WFDB Software Package. Run **make** in that directory to compile and install it if it has not been installed already.

SEE ALSO

edf2mit(1)

http://www.edfplus.info/spscs/edfplus.html Full specification of EDF+, by Bob Kemp and Jesus Olivan.

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/convert/rdedfann.c

NAME

rdsamp – read WFDB signal files

SYNOPSIS

rdsamp -r *record* [*options ...*]

DESCRIPTION

rdsamp reads signal files for the specified *record* and writes the samples as decimal numbers on the standard output. If no *options* are provided, **rdsamp** starts at the beginning of the record and prints all samples. By default, each line of output contains the sample number and samples from each signal, beginning with channel 0, separated by tabs.

Options include:

-c Produce output in CSV (comma-separated value) format (default: write output in tab-separated columns).

-f *time* Begin at the specified *time*. By default, **rdsamp** starts at the beginning of the record.

-h Print a usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-l *interval*
Limit the amount of output to the specified time *interval* (in standard time format; default: no limit). If both **-l** and **-t** are used, **rdsamp** stops at the earlier of the two limits.

-p Print times in seconds and milliseconds, and values in physical units. By default, **rdsamp** prints times in sample intervals and values in A/D units.

-P Same as **-p**, but yields higher precision in the sample values (8 decimal places rather than 3).

A single character can be attached to either **-p** or **-P** to choose the format for the printed times in the first column of output. The choices are:

-pd (or **-Pd**)
Print time of day and date if known, as [hh:mm:ss DD/MM/YYYY]. The base time and date must appear in the header file for the record; otherwise, this format is equivalent to "e" format (below).

-pe (or **-Pe**)
Print the elapsed time from the beginning of the record, as hh:mm:ss.

-ph (or **-Ph**)
Print the elapsed time in hours.

-pm (or **-Pm**)
Print the elapsed time in minutes.

-ps (or **-Ps**)
Print the elapsed time in seconds. This is the default format when using **-p** or **-P**.

-pS (or **-PS**)
Print the elapsed time in sample intervals.

-s *signal-list*
Print only the signals named in the *signal-list* (one or more input signal numbers or names, separated by spaces; default: print all signals). This option may be used to re-order or duplicate signals.

-S *signal*
Search for the first valid sample of the specified *signal* (a signal name or number) at or following the time specified with **-f** (or the beginning of the record if the **-f** option is not present), and begin printing at that time.

-t *time* Stop at the specified *time*. By default, **rdsamp** stops at the end of the record.

-v Print column headings (signal names on the first line, units on the second). The names of some signals are too wide to fit in the columns; such names are shortened by omitting the initial characters (since names of related signals often differ only at the end, this helps to make the columns identifiable). Names of units are shortened when necessary by omitting the final characters, since the initial characters are usually most important for distinguishing different units.

-X Produce output in WFDB-XML format (same as the CSV format produced using the **-c** option, but wrapped within an XML header and trailer). This format is recognized and parsed automatically by **wrsamp**.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

AVAILABILITY

This program is provided in the *app* directory of the WFDB Software Package. Run **make** in that directory to compile and install it if it have not been installed already.

The PhysioNet ATM (http://physionet.org/cgi-bin/ATM) provides web access to **rdsamp** (select **Show samples as text** from the Toolbox).

SEE ALSO

rdann(1), **setwfdb**(1), **wrsamp**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/rdsamp.c

NAME

rxr – ANSI/AAMI-standard run-by-run annotation comparator

SYNOPSIS

rxr -r *record* **-a** *reference-annotator test-annotator* [*options ...*]

DESCRIPTION

Using options **-C**, **-L**, or **-S**, *rxr* implements the run-by-run comparison algorithms described in ANSI/AAMI EC38:1998, the American National Standard for Ambulatory ECGs, and in ANSI/AAMI EC57:1998, the American National Standard for Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms. **rxr** is the reference implementation of these algorithms, and must be used to obtain the run-by-run performance statistics cited in EC38 and EC57 in order to be in compliance with the standards (see EC38, section 5.2.14, and EC57, section 4.2).

Input to this program consists of two annotation files associated with the same *record*. One of these is designated the *reference* annotation file, the other the *test* annotation file (called the 'algorithm' annotation file in EC38 and in EC57).

Options include:

-c *file* Append condensed reports to *file*.

-C *file* As for **-c**, but report SVE run statistics also.

-f *time* Begin the comparison at the specified *time* (default: 5 minutes after the beginning of the record).

-h Print a usage summary.

-l *file* Append line-format reports (EC57 Table A.7 format) to *file* (see below).

-L *file file2*
As for **-l**, but report SVE run statistics in *file2*.

-s *file* Append standard reports (EC38 section 5.2.14, EC57 Tables 7, 8 format) to *file*.

-S *file* As for **-s**, but report SVE run statistics also.

-t *time* Stop the comparison at the specified *time* (default: the end of the record if it is defined, the end of the reference annotation file otherwise; if *time* is 0, the comparison ends when the end of either annotation file is reached).

-v Verbose mode (list all discrepancies; see below).

-w *time* Set the *match window* (default: 0.15 seconds; see below).

At most one of **-c**, **-C**, **-l**, **-L**, **-s**, and **-S** can be given as an option. If '**-**' is given as a *file* argument, reports are written on the standard output. If no options are specified, *rxr* writes standard reports on the standard output (equivalent to using the option **-s -**). The output generated by selecting **-l** or **-L** includes column headings only if a *file* other than '**-**' is specified, and only if the specified *file* does not already exist. In this way, *rxr* can be used repeatedly to build up a line-format table for multiple records, for further processing by **sumstats**(1).

The **-v** option specifies that each mismatch is described on the standard output in a format similar to:

3/5(120188-121065)

where the first number is the reference run length, the second is the test run length (each of these is between 0 and 6), and the numbers in parentheses indicate the location of the match window in sample intervals.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

BUGS

Since **rxr** performs multiple passes over its input files, it cannot be used at the end of a pipe.

SEE ALSO

bxb(1), **ecgeval**(1), **epicmp**(1), **mxm**(1), **setwfdb**(1), **sumstats**(1)

Evaluating ECG Analyzers (in the *WFDB Applications Guide*)

Ambulatory Electrocardiographs (ANSI/AAMI EC38:1998)
Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms (ANSI/AAMI EC57:1998)

The last two of these publications are available from AAMI, 1110 N Glebe Road, Suite 220, Arlington, VA 22201 USA (http://www.aami.org/).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/rxr.c

NAME

sampfreq – show sampling frequency for a record

SYNOPSIS

sampfreq [*option*] *record*

DESCRIPTION

This program shows the sampling frequency (in samples per second per signal) for the specified *record*. A record may contain multiple signals sampled at different frequencies; in this case, the signals are stored in *frames* each containing at least one sample of each signal, and (by default, if no *option* is specified) **sampfreq** shows the number of frames per second. This is the number of samples per second returned for each signal when the record is read using the WFDB library function **getvec** in standard (low-resolution) mode; see **rdsamp**(1), for example.

If specified, the *option* may be one of:

-a List all signals in the record and their respective sampling frequencies.

-h Print a usage summary.

-H Show the highest frequency used for any signal in a multi-frequency record. This is the number of samples per second returned by **getvec** when the record is read in high-resolution mode; see **rdsamp**(1), for example.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

record.hea header file

SEE ALSO

rdsamp(1), **setwfdb**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/sampfreq.c

NAME

setwfdb, cshsetwfdb – set WFDB environment variables

SYNOPSIS

. setwfdb
source cshsetwfdb
call setwfdb

DESCRIPTION

WFDB applications search for input files by looking for them in an ordered list of locations called the *WFDB path*. These locations can be given by directory names or (if the WFDB library has been installed with **NETFILES** support) URLs. If the **WFDB** environment variable is set, its value specifies the WFDB path; otherwise, applications use the builtin default path specified at the time the WFDB library was compiled. The default path (**DEFWFDB**, defined in the WFDB library source file **wfdblib.h**) includes the current directory (".."), the system-wide database directory installed as part of the WFDB Software Package (usually **/usr/database**), and the PhysioBank data archive (http://www.physionet.org/physiobank/database).

WFDB applications that need access to the signal calibration database find it in a file located on the WFDB path. If the **WFDBCAL** environment variable has been set, its value specifies the name of the calibration file; otherwise, applications look for the default calibration file, the name of which (**wfdbcal**) is compiled into the WFDB library.

Many users will not need to change the defaults, but for those who do, the scripts described here may be helpful. *Important:* these programs must be customized before using them for the first time on a new machine. Since they are text files, use any text editor to customize them.

sh, **bash**, and **ksh** users:

setwfdb sets the environment variables **WFDB** and **WFDBCAL**. It must be executed using the "." as shown above. It may be convenient to include an invocation of **setwfdb** in your **.profile** file.

csh and **tcsh** users:

cshsetwfdb sets **WFDB** and **WFDBCAL** similarly for the C-shell. It must be executed using "source" as shown above. It may be convenient to include this command in your **.cshrc** file.

ENVIRONMENT

WFDB The database path: a list of directories that contain database files. An empty component is taken to refer to the current directory. All applications built with the **wfdb**(3) library search for their database input files in the order specified by **WFDB**. If **WFDB** is not set, searches are limited to the builtin WFDB path (see above). Under Unix, directory names are separated by colons (:), and the format of **WFDB** is that of the Bourne shell's **PATH** variable (see **sh**(1)). Under MS-DOS, directory names are separated by semicolons (;), and the format of **WFDB** is that of the MS-DOS **PATH** variable (colons may be used following drive specifiers within **WFDB** in this case). MacOS does not support environment variables as such; under MacOS, the builtin WFDB path is defined in **fdblib.h** as described above, and it contains a semicolon-delimited list of directories (folders) as under MS-DOS, but with colons used as directory separators rather than backslashes as under MS-DOS. Alternatively, whitespace can be used (under any environment) to separate components of the WFDB path. Under any environment, if the value of **WFDB** begins with '@', the remainder of the string is taken as the name of an "indirect WFDB path file" that defines the database path in the format described above.

This feature was introduced in WFDB library version 8.0, mainly to permit MacOS users to modify the WFDB path without recompiling the WFDB library, but it is also useful under MS-DOS to avoid the 128-character limit on the length of environment variables. Indirect WFDB path files can be nested up to 10 levels deep.

WFDBCAL

The name of the WFDB calibration file (see **wfdbcal**(5)). The usual rules for finding WFDB files by searching the WFDB path apply to the WFDB calibration file, so the value of **WFDBCAL** need not be an absolute path name. The WFDB calibration file is used by WFDB applications that need to plot signals at standard scales, as well as by **calsig**(1), which can determine the baseline

and gain of signals if calibration pulses are present and if the parameters of the calibration pulses are described in the calibration file. If **WFDBCAL** is not set by the user, the WFDB library uses a default WFDB calibration file (**wfdbcal**, named in **wfdblib.h**). If the WFDB calibration file is not readable, programs that rely on it may not choose appropriate scales for some types of signals.

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

Note that these are *templates* and will need to be customized before use:
http://www.physionet.org/physiotools/wfdb/app/setwfdb
http://www.physionet.org/physiotools/wfdb/app/cshsetwfdb

NAME

sigamp – measure signal amplitudes of a WFDB record

SYNOPSIS

sigamp -r *record* [*options ...*]

DESCRIPTION

sigamp measures either baseline-corrected RMS amplitudes or (for suitably annotated ECG signals) normal QRS peak-to-peak amplitudes for all signals of the specified *record*. It makes up to 300 measurements (but see **-n** below) for each signal and calculates trimmed means (by discarding the largest and smallest 5% of the measurements and taking the mean of the remaining 90%).

Options include:

-a *annotator*
: Measure QRS peak-to-peak amplitudes based on normal QRS annotations from the specified *annotator*.

-d *dt1 dt2*
: Set the measurement window relative to QRS annotations. Defaults: *dt1* = 0.05 (seconds before the annotation); *dt2* = 0.05 (seconds after the annotation).

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-n *nmax*
: Make up to *nmax* measurements on each signal (default: 300).

-p Print results in physical units (default: ADC units). **-p** may be followed by a single character to specifify a time format (used with **-q** and **-v** when printing individual measurements); choices are **-pd** (time of day and date if known), **-pe** (elapsed time in hours, minutes, and seconds), **-ph** (elapsed time in hours), **-pm** (elapsed time in minutes), **-ps** (elapsed time in seconds (default)), **-pS** (elapsed time in sample intervals).

-q Quick mode: print individual measurements only.

-t *time* Process until the specified *time* in *record* (default: the end of the record). Processing will be terminated prematurely if 250 measurements are made before reaching the specified *time*.

-v Verbose mode: print individual measurements as well as trimmed means.

-w *dtw* Set RMS amplitude measurement window. Default: *dtw* = 1 (second).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

calsig(1), **setwfdb**(1), **sigavg**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/sigamp.c

NAME

sigavg – calculate averages of annotated waveforms

SYNOPSIS

sigavg -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

A common problem in signal processing is to determine the shape of a recurring waveform in the presence of noise. If the waveform recurs periodically (for example, once per second) the signal can be divided into segments of an appropriate length (one second in this example), and the segments can be averaged to reduce the amplitude of any noise that is uncorrelated with the signal. Typically, noise is reduced by a factor of the square root of the number of segments included in the average. For physiologic signals, the waveforms of interest are usually not strictly periodic, however. **sigavg** averages such waveforms by defining segments (averaging windows) relative to the locations of waveform annotations.

sigavg requires a WFDB *record* containing any number of signals to be averaged, and an annotation file containing markers (fiducial points) that define a fixed point in the averaging window for each waveform. By default, all QRS (beat) annotations for the specified *annotator* are included in an average that begins 50 ms before the annotation and ends 50 ms after the annotation. The output is in text form, with times (in seconds, relative to the annotations) of each sample in the first column, and averages for each signal in the remaining columns.

Options include:

-d *dt1 dt2*
: Set the measurement window relative to QRS annotations. Negative values correspond to offsets that precede the annotations. Defaults: *dt1* = -0.05 seconds; *dt2* = 0.05 seconds.

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a usage summary.

-H Read multifrequency records in high resolution mode (default: use low resolution mode).

-p *type* [*type ...*]
: Include annotations of the specified *type*s only (default: include all QRS annotations).

-t *time* Process until the specified *time* in *record* (default: the end of the record).

-v Verbose mode: print column headings above measurements.

-z Set the baseline to zero before averaging.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

calsig(1), **setwfdb**(1), **sigamp**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/sigavg.c

NAME

signame – print names of signals of a WFDB record

SYNOPSIS

signame -r *record* [*options* ...]

DESCRIPTION

signame prints the names of the signals in the specified *record* (one per line). Using the **-s** option, only the names of the signals specified by signal number are printed.

Options include:

-h Print a usage summary.

-s *signal* [*signal* ...]
Print names for the specified *signal*(s) only. Signals are numbered 0, 1, 2 .... If the specified signal does not exist in *record*, **signame** outputs "[INVALID]".

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

signum(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/signame.c

NAME

signum – print signal numbers of a WFDB record having specified names

SYNOPSIS

signum -r *record* [*options* ...]

DESCRIPTION

signum prints the signal numbers in the specified *record* (one per line) corresponding to the specified signal names.

Options include:

-h Print a usage summary.

-s *name* [*name* ...]
Print signal numbers of signals that have the specified *name*s. Signals are numbered 0, 1, 2 .... If the specified signal does not exist in *record*, **signame** outputs "X".

If two or more signals in a record match a specified *name*, **signum** outputs the numbers of all the matching signals on a single line.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

signame(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/signum.c

NAME

skewedit – edit skew fields of header file(s)

SYNOPSIS

skewedit *record skew0* [*skew1 ... skewN*]

DESCRIPTION

This program reads the **header**(5) file for the specified *record*, changes the skew fields to match the *skew0*, *skew1*, etc. arguments, and rewrites the header file as *record*.hea in the current directory. *skew0* is the skew in samples for signal 0, *skew1* is the skew for signal 1, etc. Skews may not be negative; any omitted skews are taken to be zero.

Skew refers to time differences between samples having the same sample number in different signals. Skew may arise while digitizing multitrack analog tape recordings, for example, as a result of differences in the azimuth of the recording and playback heads of the recording equipment. It may be possible to measure skew (for example, by applying test signals simultaneously or at known intervals to all input channels, and then by measurement of the digitized test signals). When this is possible, *skewedit* can then be used to record the skew measurements in the header file.

For example, assume that a test signal applied simultaneously to all inputs of record *abc* is determined to appear on signal 0 at sample 30, on signal 1 at sample 28, on signal 2 at sample 28, and on signal 3 at sample 26. In this case, *skew0* is 4 (30 - 26), *skew1* and *skew2* are each 2, and *skew3* is 0. The command

skewedit abc 4 2 2 0

would apply the proper correction to the header file for the record. (The final '0' may be omitted from the command.)

Applications built using the WFDB library (version 9.2 or later) are able to correct for skew (the skew correction is performed by the WFDB library and is not visible to the application program). Note that skew correction does not require rewriting the signal file(s).

If you wish to create skew-corrected signal files (for example, to use with applications built using earlier versions of the WFDB library), use **xform**(1) to do so, using the header file generated by **skewedit** as input to **xform**. Note, however, that older applications can generally be updated without source changes simply by recompiling them and linking them with the current WFDB library.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

setwfdb(1), **xform**(1), **header**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/skewedit.c

NAME

snip – copy an excerpt of a WFDB record

SYNOPSIS

snip -i *input-record* **-n** *new-record* [*options*]

DESCRIPTION

snip copies the signal files (and, optionally, annotation files) of the specified *input-record*, and generates a header file, thereby creating the specified *new-record*. **snip** is usually used to extract an excerpt of its *input-record*, using the **-f** and **-t** options (see below) to specify the segment to be copied.

The program **xform**(1) can also perform this task, but offers additional flexibility (it can scale the signals, resample them at a different frequency, rearrange them, select subsets of them, or reformat them); **snip** is faster than **xform**, however.

Options are:

-a *annotator*
: Copy the specified *annotator* as well as the signal files. Two or more *annotator* arguments, separated by spaces, can follow **-a**. An annotator supplied via the standard input may be specified using '**-**', but only immediately after **-a**; in this case only, annotations are copied to the standard output.

-f *time* Begin at the specified *time* in the input record (default: the beginning of the record).

-h Print a usage summary.

-l *duration*
: Snip a segment of the specified *duration* (hh:mm:ss or s*nnnn*; overrides **-t** if given).

-m Preserve segments of multi-segment input, if possible.

-O *format*
: Write output in the specified *format*. See **header**(5) for a list of available formats (16, 80, 212, ...). If this option is omitted, **snip** uses a format that best fits the ADC resolution of the samples.

-s Suppress output of info strings in the output header file.

-t *time* Process until the specified *time* in the input record (default: continue to the end of the record).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

new-record.annotator	output annotation file
new-record.dat	output signal file
new-record.hea	output header file

SEE ALSO

setwfdb(1), **xform**(1), **header**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/snip.c

NAME

sortann – rearrange annotations in canonical order

SYNOPSIS

sortann -r *record* **-a** *annotator* [*options* ...]

DESCRIPTION

Applications that use the WFDB library (version 9.7 and later versions) may write annotations in any order. Most applications that read annotations, however, expect to find them in *time* order (with simultaneous annotations ordered by their *num* and *chan* attributes).

sortann rewrites the annotation file specified by *record* and *annotator*, arranging its contents in canonical (*time*, *num*, and *chan*) order. By default, WFDB applications run **sortann** as needed (from within *wfdbquit* or *oannclose*). If the environment variable **WFDBNOSORT** has been set (to any value), **sortann** will not be run automatically, and a warning message will be printed instead. In most such cases, you should run **sortann** as instructed by the warning message before reading the annotation file with any other WFDB application.

If the input contains two or more annotations with the same *time*, *num*, and *chan* fields, only the last one is copied. As a special case of this policy, if the last such annotation has *anntyp* = 0 (**NOTQRS**), no annotation is written at that location. Thus a program that generates input for **sortann** can effectively delete a previously written annotation by writing a **NOTQRS** annotation at the same location.

The sorted (output) annotation file is always written to the current directory. If the input annotation file is in the current directory, **sortann** replaces it unless you specify a different output annotator name (using the **-o** option). Note that the output annotation file is likely to be slightly shorter than the input file, since more compact storage is usually possible when all annotations are sorted.

If the input annotations are already in the correct order, no output is written unless you have used the **-o** option.

If you attempt to sort a very large annotation file, **sortann** may run out of memory. If this happens, use the **-f** and **-t** options to work on the file in sections of any convenient size, one at a time, then use **mrgann**(1) to concatenate the sections. Note that you must specify an output annotator name (with **-o**) when using the **-f** or **-t** options (to avoid replacing the entire input file with a sorted subset of its contents).

The working memory required by **sortann** is approximately 10 times the size of the annotation file. Since annotation files are rarely as large as 1 megabyte and available memory is rarely less than 10 megabytes, it is unlikely that **sortann** will exhaust available memory, however.

Options include:

-f *time* Begin at the specified *time*. By default, **sortann** starts at the beginning of the record.

-h Print a usage summary.

-o *output-annotator*
Write output to the annotation file specified by *output-annotator* and (as specified using **-r**) *record*. By default, **sortann** replaces the input annotation file.

-t *time* Stop at the specified *time*.

The **-f** and **-t** options may be used to select a portion of an annotation file for processing.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

mrgann(1), **setwfdb**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/sortann.c

NAME

sqrs, sqrs125 – single-channel QRS detector

SYNOPSIS

sqrs -r *record* [*options ...*]
sqrs125 -r *record* [*options ...*]

DESCRIPTION

sqrs attempts to locate QRS complexes in an ECG signal in the specified *record*. The detector algorithm is based on example 10 in the *WFDB Programmer's Guide*, which in turn is based on a Pascal program written by W.A.H. Engelse and C. Zeelenberg, ''A single scan algorithm for QRS-detection and feature extraction'', *Computers in Cardiology* **6**:37-42 (1979). **sqrs** does not include the feature extraction capability of the Pascal program. The output of **sqrs** is an annotation file (with annotator name **qrs**) in which all detected beats are labelled normal; the annotation file may also contain 'artifact' annotations at locations that **sqrs** believes are noise-corrupted.

sqrs can process records containing any number of signals, but it uses only one signal for QRS detection (signal 0 by default; this can be changed using the **-s** option, see below). **sqrs** is optimized for use with adult human ECGs. For other ECGs, it may be necessary to experiment with the sampling frequency as recorded in the input record's header file (see **header**(5)) and the time constants indicated in the source file.

sqrs uses the WFDB library's *setifreq* function to resample the input signal at 250 Hz if a significantly different sampling frequency is indicated in the header file. **sqrs125** is identical to *sqrs* except that its filter and time constants have been designed for 125 Hz input, so that its speed is roughly twice that of **sqrs**. If the input signal has been sampled at a frequency near 125 Hz, the quality of the outputs of **sqrs** and **sqrs125** will be nearly identical. (Note that older versions of these programs did not resample their inputs; rather, they warned if the sampling frequency was significantly different than the ideal frequency, and suggested using **xform**(1) to resample the input.)

This program is provided as an example only, and is not intended for any clinical application. At the time the algorithm was originally published, its performance was typical of state-of-the-art QRS detectors. Recent designs, particularly those that can analyze two or more input signals, may exhibit significantly better performance.

Options include:

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-m *threshold*
Specify the detection *threshold* (default: 500 units); use higher values to reduce false detections, or lower values to reduce the number of missed beats.

-s *signal*
Specify the *signal* (number or name) to be used for QRS detection (default: 0).

-t *time* Process until the specified *time* in *record* (default: the end of the *record*).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

EXAMPLES

To mark QRS complexes in record 100 beginning 5 minutes from the start, ending 10 minutes and 35 seconds from the start, and using signal 1, use the command:

sqrs -r 100 -f 5:0 -t 10:35 -s 1

The output annotations may be read using (for example):

rdann -a qrs -r 100

To evaluate the performance of this program, run it on the entire record, by:

sqrs -r 100

and then compare its output with the reference annotations by:

bxb -r 100 -a atr qrs

SEE ALSO

bxb(1), **rdann**(1), **setwfdb**(1), **wqrs**(1), **xform**(1)

AUTHORS

George B. Moody (george@mit.edu). This program is a fairly literal translation with minor corrections of the Pascal original by WAH Engelse and Cees Zeelenberg.

SOURCE

http://www.physionet.org/physiotools/wfdb/app/sqrs.c
http://www.physionet.org/physiotools/wfdb/app/sqrs125.c

NAME

stepdet – single-channel step change detector

SYNOPSIS

stepdet -r *record* [*options ...*]

DESCRIPTION

This program analyzes one signal of a PhysioBank-compatible *record*, detecting and annotating rising and falling step changes. Typically this can be useful for finding transitions in a recorded digital stimulus or event marker signal, especially if the signal is noise-contaminated (as may occur if it has been recorded via an analog-to-digital converter).

Options include:

-a *annotator*
: Write annotations to the specified *annotator* (default: 'steps')

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a brief usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-m *tup tdown*
: Specify thresholds for transitions from low to high (*tup*, default: 550) and from high to low (*tdown*, default: 450).

-s *signal*
: Specify the *signal* (number or name) to be used for step detection (default: 0).

-t *time* Process until the specified *time* in *record* (default: the end of the *record*).

tup is the threshold for detecting a rising step change (annotated as 'R'), and *tdown* is the threshold for detecting a falling ('F') step change. This program requires that *tup* > *tdown*. Using its -m option, set *tup* to a value significantly greater than *tdown* to avoid false detections of transitions due to noise in the signal. Noise spikes that still cause false detections can often be avoided by median-filtering the signal (see mfilt(1)) before using it as input to this program.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

mfilt(1)

AUTHORS

George B. Moody (george@mit.edu).

SOURCE

http://www.physionet.org/physiotools/wfdb/app/stepdet.c

NAME

sumann – summarize the contents of a WFDB annotation file

SYNOPSIS

sumann -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

sumann reads the annotation file specified by *record* and *annotator* and produces a tabular summary of its contents, including the number of annotations of each type as well the duration and number of episodes of each rhythm and signal quality.

Options include:

-f *time* Begin at the specified *time*.

-h Print a usage summary.

-o *beat-table rhythm-table*
Append summaries of beat and rhythm annotations to the specified *beat-table* and *rhythm-table* files. The summaries for the specified *record* are written as a single line in CSV (comma-separated value) format in each file. If either file does not exist, it is created, and a header line containing column names is written to it. If the **-o** option is omitted, a more easily readable summary is written to the standard output instead.

-q Summarize QRS annotations only.

-t *time* Stop at the specified *time*.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

rdann(1), **setwfdb**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/sumann.c

NAME

sumstats – derive aggregate statistics from bxb, rxr, etc., line-format output

SYNOPSIS

sumstats *file*

DESCRIPTION

This program derives the aggregate statistics described in sections 3.5.2 and 3.5.3 of the American National Standard, *Testing and reporting performance results of cardiac rhythm and ST segment measurement algorithms* (ANSI/AAMI EC57:1998, based on the earlier AAMI ECAR:1987), and in sections 4.2.14.4.1 and 4.2.14.4.2 of the American National Standard, *Ambulatory electrocardiographs* (ANSI/AAMI EC38:1998).

To use this program, first generate a line-format report *file* using the **-l** or **-L** options of **bxb**(1), **epicmp**(1), **mxm**(1), or **rxr**(1). This file must include column headings so that **sumstats** can recognize the file type. Output is written to the standard output; it includes a copy of the input file, with aggregate statistics appended at the end.

SEE ALSO

bxb(1), **ecgeval**(1), **epicmp**(1), **mxm**(1), **plotstm**(1), **rxr**(1)

Evaluating ECG Analyzers (in the *WFDB Applications Guide*)

American National Standard ANSI/AAMI EC38:1998, Ambulatory Electrocardiographs

Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms (publication AAMI EC57:1998)

The last two publications are available from AAMI, 1110 N Glebe Road, Suite 220, Arlington, VA 22201 USA (http://www.aami.org/).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/sumstats.c

NAME

tach – heart rate tachometer

SYNOPSIS

tach -r *record* **-a** *annotator* [*options ...*]

DESCRIPTION

tach reads an annotation file (specified by the *annotator* and *record* arguments) and produces a uniformly sampled and smoothed instantaneous heart rate signal. Smoothing is accomplished by finding the number of fractional R-R intervals within a window (with a width of $2k$ output sample intervals, where k is a smoothing constant) centered on the current output sample. By default, the output is in text form, and consists of a column of numbers, which are samples of the instantaneous heart rate signal (in units of beats per minute). Optionally, the output sample number can be printed before each output sample value. Alternatively, **tach** can create a WFDB record containing the heart rate signal.

Studies of heart rate variability generally require special treatment of ectopic beats. Typically, ventricular ectopic beat annotations are removed from the input annotation file and replaced by 'phantom' beat annotations at the expected locations of sinus beats. The same procedure can be used to fill in gaps resulting from other causes, such as momentary signal loss. It is often necessary to post-process the output of *tach* to remove impulse noise in the heart rate signal introduced by the presence of non-compensated ectopic beats, especially supraventricular ectopic beats. Note that **tach** performs none of these manipulations, although it usually attempts limited outlier rejection (**tach** maintains an estimate of the mean absolute deviation of its output, and replaces any output that is more than three times this amount from the previous value with the previous value).

Options include:

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-F frequency
Produce output at the specified sampling frequency (default: 2 Hz).

-h Print a usage summary.

-i *rate* For outlier detection, assume an initial rate of *rate* bpm (default: 80).

-l *duration*
Process the record for the specified *duration*, beginning at the time specified by a previous **-f** option, or at the beginning of the record.

-n *n* Produce exactly *n* output samples, adjusting the output frequency so that they are evenly spaced throughout the interval specified by previous **-f** and **-t** or **-l** options. This option is particularly useful if the output of **tach** is to be used as input for a fast Fourier transform, since *n* can be chosen to be a convenient power of two.

-o *record*
Write output to signal and header files for the specified *record* (which should not be the same as the input record). This option suppresses the standard text output of **tach**.

-O Disable outlier rejection.

-s *k* Set the smoothing constant to *k* (default: 1; *k* must be positive).

-t *time* Process until the specified *time* in *record* (default: the end of the *record*).

-v Print the output sample number before each output sample value.

-V, **-Vs**, **-Vm**, **-Vh**
Print the output sample time in seconds (using **-V** or **-Vs**), minutes (using **-Vm**), or hours (using **-Vh**) before each output sample value. Only one of these options can be used at a time.

Reference ('atr') annotation files can be used as input to **tach**, but files that contain manually-inserted annotations are less suitable, since annotation placement is likely to be less consistent than in annotation files generated by programs such as **sqrs**(1).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

DIAGNOSTICS

annotation buffer overflow

Use a smaller smoothing constant, a higher output frequency, or recompile *tach* with a larger value for **ABL**.

SEE ALSO

setwfdb(1), **sqrs**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/tach.c

NAME

time2sec – convert WFDB standard time format into seconds

SYNOPSIS

time2sec [**-r** *record*] *time*

DESCRIPTION

This program converts the specified time interval, *time* (in WFDB standard time format), into seconds. Enclose TIME in square brackets (e.g., [9:0:0]) to convert a time of day to the elapsed time in seconds from the beginning of the (optionally) specified RECORD.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

record.hea	header file

SEE ALSO

setwfdb(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/time2sec.c

NAME

wabp – arterial blood pressure (ABP) pulse detector

SYNOPSIS

wabp -r *record* [*options ...*]

DESCRIPTION

wabp attempts to locate arterial blood pressure (ABP) pulse waveforms in a continuous ABP signal in the specified *record*. The detector algorithm is based on analysis of the first derivative of the ABP waveform. The output of **wabp** is an annotation file (with annotator name **wabp**) in which all detected beats are labelled normal.

wabp can process records containing any number of signals, but it uses only one signal for ABP pulse detection (by default, the lowest-numbered ABP, ART, or BP signal; this can be changed using the **-s** option, see below). **wabp** is optimized for use with adult human ABPs. It has been designed and tested to work best on signals sampled at 125 Hz. For other ABPs, it may be necessary to experiment with the sampling frequency as recorded in the input record's header file (see **header**(5)).

wabp optionally uses the WFDB library's *setifreq* function to resample the input signal at 125 Hz.

Options include:

-d Dump the raw and pre-processed input samples in text format on the standard output, but do not detect or annotate ABP pulses.

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a brief usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-R Resample the input at 125 Hz (default: do not resample).

-s *signal*
Specify the *signal* (number or name) to be used for ABP pulse detection (default: the lowest-numbered ABP, ART, or BP signal).

-t *time* Process until the specified *time* in *record* (default: the end of the *record*).

-v Verbose mode: print information about the detector parameters.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

EXAMPLES

To mark ABP pulses in record slp60 of the slpdb database, beginning 5 minutes from the start, ending 10 minutes and 35 seconds from the start, and using signal 1, use the command:

wabp -r slpdb/slp60 -f 5:0 -t 10:35 -s 1

The output annotations may be read using (for example):

rdann -r slpdb/slp60 -a wabp

SEE ALSO

bxb(1), **ecgpuwave**(1), **rdann**(1), **setwfdb**(1), **sqrs**(1), **wqrs**(1)

Zong W, Heldt T, Moody GB, and Mark RG. An open-source algorithm to detect onset of arterial blood pressure pulses. *Computers in Cardiology* **30**:259–262 (2003).

AUTHORS

Wei Zong (wzong@mit.edu) and George B. Moody (george@mit.edu).

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wabp.c

NAME

wav2mit, mit2wav – convert between .wav and WFDB-compatible formats

SYNOPSIS

wav2mit -i *file***.wav** [*options ...*]
mit2wav -o *file***.wav -r** *record* [*options ...*]

DESCRIPTION

These programs convert files in the widely-used **.wav** audio file format into WFDB format files (as used in PhysioBank) and vice versa. Most **.wav** files are already written in a WFDB-compatible format, although the reverse is not true. (An embedded header is required by **.wav** format, and is allowed but is not usually present in WFDB-format signal files.)

wav2mit creates a WFDB record from *file***.wav**. If the input file is written in an MIT-compatible signal file format, all that is required in this case is to create a suitable WFDB-format **.hea** header file that describes the **.wav** file's format. Some **.wav** files are written using variants of the format that are not readable by the WFDB library; the current version of **wav2mit** does not attempt to convert such files, but warns that they are not compatible. Options for **wav2mit** include:

-h Print a brief usage summary.

-r *record*
Create the specified *record* (default: use the base name of the input file as the record name).

mit2wav reads the specified WFDB-format *record* (header and signal files) and creates a **.wav** file containing the same data. Note that much of the data description contained in the WFDB-format header file cannot be preserved in the **.wav** file. Options for **mit2wav** include:

-h Print a brief usage summary.

-n *record*
Create a header file for the output (**.wav**) signal file, so that it can be read by WFDB applications as the specified *record*.

It may be possible to create analog signals by playing the .wav file through a sound card, but you should be aware of the following potential pitfalls:

Your sound card, and the software that comes with it, may not be able to play **.wav** files containing three or more signals. If this is a problem, you will need to extract one or two signals to include in the **.wav** file from your original recording (for example, using **xform**(1)).

Your sound card and its software may be unable to play **.wav** files at other than certain fixed sampling frequencies (typically 11025, 22050, and 44100 Hz). If this is a problem, you will need to resample the input at one of the frequencies supported by your sound card (for example, using **xform**(1)) before converting it to **.wav** format using this program.

Your sound card may not be able to reproduce the frequencies present in the input. This is *very* likely if you are trying to recreate physiologic signals such as ECGs (with most of the useful information in the 0.1 to 30 Hz band) using a consumer sound card (which probably does not reproduce frequencies below the lower limit of human hearing (around 30 Hz). One possible solution to this problem is to create a digital signal containing a higher-frequency carrier signal, amplitude-modulated by the signal of interest, and to convert this AM signal into a **.wav** file; on playback, an analog AM demodulator would then recover the original low-frequency signal of interest. If you successfully implement this solution, please send details to the author.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

AVAILABILITY

These programs are provided in the *convert* directory of the WFDB Software Package. Run **make** in that directory to compile and install them if they have not been installed already.

SEE ALSO

a2m(1), **edf2mit**, **snip**(1), **xform**(1), **wfdb**(3), **header**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/convert/wav2mit.c
http://www.physionet.org/physiotools/wfdb/convert/mit2wav.c

NAME

wave – waveform analyzer, viewer, and editor

SYNOPSIS

wave -r *record*[**+***record ...*] [*options ...*]

DESCRIPTION

wave can be used to view the specified WFDB *record* or records on any display controlled by an X11 server. It includes facilities for interactive annotation editing. The keyboard and mouse are used to control the display interactively. First-time users should read the *WAVE User's Guide*, available at **http://physionet.org/physiotools/wug/** (or, while **wave** is running, choose *User's Guide* from the *Help* panel).

If you specify more than one *record*, a separate **wave** process is started for each record. Note that all records to be opened must be listed in a single command-line argument following **-r**, with **+** characters (not spaces) between the record names. See 'Running two or more WAVE processes' below.

Use the left mouse button to make selections, and the right mouse button to open menus (indicated by triangular glyphs at the right end of some buttons). See the *Guide* or choose *Annotation Editing* from the *Help* panel).

Options are:

-a *annotator*
: Open the specified annotation file for the previously specified *record* or records.

-dpi *xx*[**x***yy*]
: Calibrate **wave** for use with a display having a resolution of *xx* (by *yy*) dots per inch.

-f *time* Open the record(s) beginning at the specified *time*.

-g Use shades of grey only, even on a color monitor.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-m Use monochrome (usually black and white) only, even on a color or greyscale monitor. The line styles selected by the **-m** option may be easier to distinguish on some greyscale monitors than the default shades of grey.

-O Use overlay graphics for maximum speed and display quality if possible. This is the usual default if the X server supports a PseudoColor or GrayScale visual. This option exists only to force use of overlay graphics if a different mode has been chosen as the default.

-s *signal-list*
: Initialize the signal list. By default, the signal list includes all available signals, in numerical order.

-S Use the standard (shared) color palette, even if it is possible to modify the palette. Using this option conserves color resources if you have other applications that use non-standard colors, at the expense of some speed in redrawing the display. The **-S** option may be used in combination with the **-g** option if desired.

-V*x* Set display option *x*. See 'Display Options' below for details.

Note that **wave** queries the X server to determine the display capabilities and resolution; it is not necessary to use the **-g**, **-m**, or **-S** options unless you wish to restrict **wave**'s use of the available capabilities. Use the **-dpi** option to override the server's default resolution if it is incorrect and cannot be changed otherwise (see comments below under 'Resources').

The system on which **wave** runs (the ''host'' system) need not be the same as the system to which your keyboard, mouse and display are connected (the ''local'' system), provided only that the host and local systems are on the same network. If you wish to run **wave** remotely, simply log in to the host using **ssh**, which normally handles display redirection automatically. If you use some other method to log in remotely, such as

rlogin (not recommended) or **telnet** (not recommended), it is usually necessary to grant permission for the host system to open windows on the local system's display (generally, this is accomplished using **xhost** on the local system; see the documentation for your X server for details), and to set the **DISPLAY** environment variable on the host system appropriately (if the local system runs UNIX, the value of **DISPLAY** should be *local-hostname***:0.0** in most cases; again, consult your X server documentation).

ENVIRONMENT

wave uses many environment variables; they are listed in this section roughly in order of importance. Many of them need not be set at all, since **wave** uses reasonable default values in most cases. Those that are set must be set on the host system.

DISPLAY
The name of the X server and display you are using (see above). If you are using **wave** locally, or if you are logged in via **ssh**, **DISPLAY** should be set automatically and should not need to be changed.

WFDB The database path (see **setwfdb**(1)). If not set, **wave** can find database files only in the builtin WFDB path. If you edit annotation files and wish to reopen them later, be sure that the current directory (in which **wave** writes any edited annotation files) is the first directory in your database path.

WFDBCAL
The WFDB calibration file (see **setwfdb**(1) and **wfdbcal**(5)). If not set, **wave** reads the builtin default calibration file; if this is not accessible, **wave** may not scale signals other than ECGs correctly.

WAVEMENU
The name of the analysis menu file (see below); if not set, **wave** uses **wavemenu** if it exists in the current directory, or **$MENUDIR/wavemenu.def** otherwise.

SHELL
The command interpreter used within the Analysis Commands window; if not set, **wave** uses **/bin/sh** (the Bourne shell). Other shell-related variables, such as **PATH**, are also significant when **wave** is running commands within the Analysis Commands window.

EDITOR
The name of the text editor to be used for modifying the analysis menu file and the log file. If not set, **wave** uses **textedit** (a simple editor included with the XView toolkit).

PRINTER
The name of a printer to be used for paper output; if not set, **wave** uses the default printer.

PSPRINT
The command used to print PostScript data from the standard input; if not set, **wave** uses '**lpr -P$PRINTER**'.

TEXTPRINT
The command used to print text from the standard input; if not set, **wave** uses '**lpr -P$PRINTER**'.

ANNTAB
The name of a file that contains custom annotation definitions (see 'Resources', below, for details). If not set, **wave** uses standard annotation definitions only.

The environment variables below are not needed unless the **wave** binary distribution, or XView, has been installed in non-standard directories:

HELPPATH
The path for XView spot help; if not set, **wave** initializes it to **/usr/lib/help**. **wave**'s own spot help is in **$HELPDIR/wave**, which is appended to the end of **HELPPATH** by **wave**.

HELPDIR
The directory in which **wave**'s help directory is located; if not set, **wave** uses **/usr/local/help**.

MENUDIR
The name of the directory that contains the default analysis menu file; if not set, **wave** uses **/usr/local/lib**.

RESDIR
The name of the directory in which system-wide default X11 resource files are kept; if not set, **wave** uses **/usr/lib/X11/app-defaults**. **XUSERFILESEARCHPATH**, **XAPPLRESDIR**, and **XENVIRONMENT** are also used, together with **HOME** and **USER**, to locate resource files (see **X**(1)).

RESOURCES

You can control many aspects of **wave**'s appearance and behavior by setting its resources. If you are not familiar with this concept, refer to an introductory book on using the X Window System, such as Darwin, Quercia, and O'Reilly's *X User's Guide: Open Look Edition* (see the link below). Since **wave** is built using the XView toolkit, all of the resources listed in **xview**(7) can be used with **wave**. In addition, the following **wave**-specific resources may also be set:

Wave.AllowDottedLines
This resource specifies if **wave** is allowed to render dotted lines. **wave** normally draws annotation marker bars as dotted lines, and may use dotted lines for other display elements on black-and-white displays for clarity. Some X servers do not properly render dotted lines, however; if you observe irregular or missing annotation marker bars, change the value of this resource from **True** to **False**.

Wave.Anntab
This resource specifies the name of a file that contains a table of annotation definitions. The environment variable **ANNTAB** can also be used to specify this filename; the resource overrides the environment variable if both are set. The file contains one-line entries of the form

15 % Funny looking beat

in which the first field specifies the (numeric) annotation code in the range between 1 and **ACMAX** inclusive (see **/usr/include/wfdb/ecgcodes.h** for a list of predefined codes and for the definition of **ACMAX**); the second field ('%' in the example) is a mnemonic (used in annotation display and entry), and the remainder of the entry is a description of the intended use of the annotation code (which appears next to the mnemonic in the 'Type' field and menu of 'Annotation Template' windows). Lines in the annotation table that begin with '#' are treated as comments and ignored. It is not necessary to specify an annotation table when editing an existing annotation file unless previously undefined annotation types are to be added to it during the editing process, although it is generally harmless to do so.

Wave.Dpi
This resource specifies the display resolution in dots per inch in the form *MM***x***NN*, where *MM* is the horizontal resolution and *NN* is the vertical resolution. Normally, the resolution is known to the X server, and it is unnecessary to specify this resource. If your X server is misinformed, **wave**'s calibrated display scales will be incorrect; the best solution is to specify the resolution using a server option such as the **-dpi** option supported by MIT's X11 servers, since this will solve problems common to any other applications that require calibrated scales as well. Not all X11 servers support such an option, so this resource is available as a work-around. The command-line option **-dpi** overrides the resource if both are specified. (If you don't know the resolution, use **xdpyinfo**(1) to determine what your X server thinks it is. Then run **wave**, enable the grid display, and measure the grid squares with a ruler. If they are larger than 5 mm, the number of dots per inch returned by **xdpyinfo** is too large; adjust the **Wave.Dpi** resource proportionally, and repeat the process until the grid squares measure 5 mm in each direction.)

Wave.GraphicsMode
This resource specifies the graphics mode used by **wave**; it can be overridden using the **-g**, **-m**, **-O**, or **-S** options. The legal values are **1** (monochrome mode), **2** (overlay greyscale mode), **4**

(shared color mode), **6** (shared grey mode), and **8** (overlay color mode).

Wave.SignalWindow.{Grey|Color}.*Element*

These resources specify the colors to be used on greyscale or color displays. The 'Color.*' resources are used only if the display is color-capable and neither greyscale nor monochrome mode has been specified. The defaults are:

Element	**Grey**	**Color**
Background	white	white
Grid	grey75	grey90
Cursor	grey50	orange red
Annotation	grey25	yellow green
Signal	black	blue

Wave.SignalWindow.Mono.Background

In monochrome mode, the background is normally white, and all other display elements are normally black. The reverse can be obtained by setting this resource to **black**. (There is at least one server for which this fails.)

Wave.Scope.{Grey|Color}.{Foreground|Background}

These resources specify the colors to be used in the Scope window on greyscale or color displays. The Foreground color is used for the waveform and the time display; by default, it matches the color used for signals in the signal window (see the previous item). Some X servers do not allow the background color of the Scope window to be set, because of the color map animation and stippled erasing techniques used.

Wave.Scope.Mono.Background

This resource can be used to invert the foreground and background of the Scope window when WAVE is running in monochrome mode. This does not work for all X servers.

Wave.SignalWindow.{Height_mm|Width_mm}

These resources specify the preferred dimensions (in millimeters) for the signal window. The defaults are 120 and 250 respectively.

Wave.SignalWindow.Font

This resource specifies the font used to display annotations and time marks in the signal window. The default is **fixed**.

Wave.TextEditor

This resource specifies the name of the text editor invoked by **wave** to permit you to edit **wave**'s log and analysis menu files. The default is **textedit** (the OpenLook visual editor). You may override this resource by using the environment variable **EDITOR**, which is also used by many other UNIX applications that invoke editors.

Display options

Initial values for the settings controlled from **wave**'s View window can be specified using either X resources or command-line options. Once suitable settings have been selected, use the 'Save as new defaults' button in **wave**'s View window to record them in your **.Xdefaults** file. In this section, the X resource name is specified first, and the command-line option follows.

By default, all of the display options in the first group are off (**False**); set any of these X resources to **True** to enable these options, or use the command-line options to do so.

Wave.View.Subtype (**-Vs**)

Display annotation **subtyp** fields.

Wave.View.Chan (**-Vc**)

Display annotation **chan** fields.

Wave.View.Num (**-Vn**)
Display annotation **num** fields.

Wave.View.Aux (**-Va**)
Display annotation **aux** fields.

Wave.View.Markers (**-Vm**)
Display annotation marker bars.

Wave.View.SignalNames (**-VN**)
Display signal names along the left edge of the signal window.

Wave.View.Baselines (**-Vb**)
Display baselines for any DC-coupled signals, and label the zero levels and the units along the right edge of the signal window.

Wave.View.Level (**-Vl**)
While the pointer is in the signal window and any mouse button is depressed, track the intersections of the marker bar with the signals and draw horizontal marker bars across the signal window at the levels of these intersections.

The remaining resources and command-line display options correspond to the menu buttons in **wave**'s View window. The value of each resource, or the numeric argument that immediately follows the command-line option, should match the position of the desired menu choice, where the top item on each menu is in position 0, the one below it is in position 1, etc. For example, to set the initial amplitude scale to 5 mm/mV (the item at position 2 in the 'Amplitude scale' menu), add **-Vv 2** to the command line, or **Wave.View.AmplitudeScale:2** to the X11 resource database.

Wave.View.TimeScale (**-Vt**)
Set the time scale:

-Vt 0 0.25 mm/hour
-Vt 1 1 mm/hour
-Vt 2 5 mm/hour
-Vt 3 0.25 mm/min
-Vt 4 1 mm/min
-Vt 5 5 mm/min
-Vt 6 25 mm/min
-Vt 7 50 mm/min
-Vt 8 125 mm/min
-Vt 9 250 mm/min
-Vt 10 500 mm/min
-Vt 11 12.5 mm/sec
-Vt 12 25 mm/sec (default)
-Vt 13 50 mm/sec
-Vt 14 125 mm/sec
-Vt 15 250 mm/sec
-Vt 16 500 mm/sec
-Vt 17 1000 mm/sec
-Vt 18 2000 mm/sec
-Vt 19 5000 mm/sec
-Vt 20 10 mm/ms
-Vt 21 20 mm/ms
-Vt 22 50 mm/ms
-Vt 23 100 mm/ms
-Vt 24 200 mm/ms
-Vt 25 500 mm/ms

Wave.View.AmplitudeScale (**-Vv**)
Set the amplitude scale (0: 1 mm/mV; 1: 2.5 mm/mV; 2: 5 mm/mV; 3: 10 mm/mV (default); 4: 20 mm/mV; 5: 40 mm/mV; 6: 100 mm/mV).

Wave.View.SignalMode (**-VS**)
Set the choice on the 'Draw' menu (0: all signals (default); 1: listed signals only).

Wave.View.AnnotationMode (**-VA**)
Set the choice on the 'Show annotations' menu (0: centered (default); 1: attached to signals; 2: as a signal).

Wave.View.TimeMode (**-VT**)
Set the choice on the 'Time display' menu (0: elapsed (default); 1: absolute; 2: in sample intervals).

Wave.View.GridMode (**-VG**)
Set the choice on the 'Grid' menu (0: none; 1: 0.2 s; 2: 0.5 mV; 3: 0.2s x 0.5 mV (default)).

In addition to the usual ways of setting X resources, it is possible to set any of those listed above, as well as any of the generic XView resources, by using the **-xrm** or **-default** options on the command line when starting **wave**. For example, you can set the background color of the signal window using a command such as

wave -r 100s -xrm Wave.SignalWindow.Color.Background:lightblue

RUNNING TWO OR MORE WAVE PROCESSES

By specifying two or more record names, separated by '**+**' characters, in the command-line argument that follows '**-r**' (see above), you may open separate WAVE signal windows (processes) for each record. These processes are almost completely independent: from any signal window, you may navigate within the record, change display settings, edit annotations, run external analysis programs, quit the process, etc., without affecting any other signal windows.

For example, you may open two signal windows for the same record by:

wave -r 100+100 -a atr

You can now move about the record freely in either window. This facility makes it easy to compare different segments of the record. Note that whenever two or more windows are displaying the same set of annotations, as in this case, only one should be editing the annotations at any given time.

The window associated with the *last* record named on the command line has a special status: it is designated the master signal window, and an extra button (labelled 'Sync') appears at the top of this window. Clicking on this button causes all of the other signal windows to be redrawn so that the times shown in their lower left corners match that in the master signal window. (Note, however, that if you have quit a signal window from the middle of the list, any signal windows from earlier in the list will no longer respond to sync requests.)

By default, all command-line arguments apply to all signal windows. You may specify an argument that is to apply to only one signal window, however, by prefixing the argument with '**+***n***/**', where *n* is the signal window number. (The first signal window, corresponding to the first record named on the command line, is signal window number 0; the next is number 1, etc.)

This facility has many applications. For example, you may wish to open two copies of the same record, with two different annotators:

wave -r 100+100 -a +0/atr +1/qrs

In this case, record 100 is opened in two windows, with annotator 'atr' in window 0 and annotator 'qrs' in window 1. (The '**-a**' option applies to both windows since it does not have a '**+***n***/**' prefix.)

As another example, you may wish to discuss a record with colleagues at other locations:

**wave -r 200+200+200 -a qrs +0/-display +0/atlantic.bigu.edu:0 **
+1/-display +1/pacific.widget.com:0

Here, record 200 is opened in three windows. Window 0 is opened on display 0 of atlantic.bigu.edu,

window 1 on display 0 of pacific.widget.com, and window 2 (the master window) on the local display. (For this to work, your colleagues must first allow your computer to open windows on their displays, typically using **xhost**. See **xview**(7) for information about the **-display** option. Notice that the '***+n/***' prefix must be attached to both the '**-display**' option and to its argument in order to apply both of these arguments to the same signal window.) Your colleagues can freely move about the record, but you can direct the discussion at any time by using the Sync button in your signal window. In a case such as this one, anyone can enable editing; you should do so only after making sure that no one else has. Once you have saved your work (by selecting 'Save' from the File menu), your changes become visible to your colleagues if they reload the annotations (by clicking on 'Reload' from the Load window).

As a final example, the MIMIC Database includes both high-resolution waveform records and medium-resolution (roughly 1 sample per second) computed measurement records. You may view both of these at the same time using a command such as:

wave -r 237+237n -a all

Typically, you will wish to view the high-resolution and low-resolution data at different time scales. Although **wave** attempts to choose reasonable defaults, you can adjust the scales independently if you wish:

wave -r 237+237n -a all +1/-Vt +1/2

If you use **wavescript** or **wave-remote** to control the master signal window (this happens by default unless you use the **-pid** option of these programs to control a different signal window), the other signal windows are kept synchronized with the master window.

Note that you cannot *increase* the number of signal windows in a group once you have started a **wave** process group, although you can run more than one process group at a time if you wish.

MENU FILE

wave uses a simple menu file to allow you to set up analysis options. Each line in the file corresponds to a button in the Analyze window (except for empty lines and lines that begin with '#', which are ignored). Within each line, the syntax is *label*<tab>*action*, where <tab> is one or more tab characters. The *label* field is used to identify a command button in the Analyze window, and the *action* field is any command acceptable to your shell. *button-label* and *action* may include spaces if needed; if necessary, a '\' may be used at the end of a line to indicate that it is continued on the next line. Before the command is executed, **wave** replaces certain tokens with appropriate strings; these include:

$RECORD
The name of the current record.

$ANNOTATOR
The name of the current input annotator.

$START
The currently selected 'start analysis' time.

$END The currently selected 'end analysis' time.

$DURATION
The time interval between **$END** and **$START**.

$LEFT
The time corresponding to the left edge of the signal window.

$RIGHT
The time corresponding to the right edge of the signal window.

$WIDTH
The time interval between **$RIGHT** and **LEFT**.

$SIGNAL
The currently selected signal number (as shown in the Analyze window).

$SIGNALS
The current signal list (as shown in the Analyze window).

$LOG The name of the current log file (as shown in the Log window).

$WFDB
The WFDB path (from the Load window).

$WFDBCAL
The name of the WFDB calibration file (from the Load window).

$TSCALE
The time scale, in mm/sec.

$VSCALE
The amplitude scale, in mm/mV.

$DISPMODE
The annotation display mode (0: annotations displayed in center, no marker bars; 1: annotations displayed in center, long marker bars; 2: annotations attached to signals, no bars; 3: annotations attached to signals, short bars; 4: annotations displayed as a signal, no bars; 5: annotations displayed as a signal, long bars)

$PSPRINT
The command for printing PostScript data from the standard input, as specified in the Print Setup window.

$TEXTPRINT
The command for printing text from the standard input, as specified in the Print Setup window.

$URL The URL specified by the most recently selected link.

Other tokens that begin with '$' are passed to the shell unchanged.

Example
The default menu file includes the following lines (among others):

Mark QRS complexes	sqrs -r $RECORD -f $START -t $END -s $SIGNAL
Calibrate	calsig -r $RECORD -f $START -t $END -s $SIGNALS
Extract segment	snip -i $RECORD -f $START -t $END -n n_$RECORD \ -a $ANNOTATOR
List annotations	rdann -r $RECORD -a $ANNOTATOR -f $START -t $END
List samples	rdsamp -r $RECORD -f $START -t $END -s $SIGNALS
Print chart	echo $RECORD $START-$END | \ pschart -a $ANNOTATOR -g -l -R -s $SIGNALS - | $PSPRINT
Print full disclosure	echo $RECORD $START-$END | \ psfd -a $ANNOTATOR -g -l -R -s $SIGNALS - | $PSPRINT

KEYBOARD COMMANDS

Whenever the pointer is in the signal window, the normal arrow pointer is replaced by a crosshair pointer. At these times, the numeric keypad and several of the function keys may be used for many annotation editing and display operations, and the normal alphanumeric and punctuation keys can be used to select single-character annotation mnemonics (displayed in the Annotation Template window). 'Num Lock' must be off if you wish to use the keypad for editing operations. Some of the function and numeric keypad commands work on Sun keyboards only; in these cases, alternate keyboard commands for use with PC and other keyboards are shown in parentheses. Most of these alternate commands also work on Sun keyboards.

<Help> (*<F1>*)
Open XView spot help for the item under the pointer. (Unlike most of the other keyboard commands, this command is available at any time, not only when the pointer is in the signal window.)

<left arrow>
Select the annotation to the left of the pointer. (Click left to do this using the mouse. These actions also work when the pointer is in the scope window.)

<right arrow>
Select the annotation to the right of the pointer. (Click right to do this using the mouse. These actions also work when the pointer is in the scope window.)

<up arrow> Move the selected annotation up one signal (i.e.,
decrement its **chan** field). This command works in multi-edit mode only (enter multi-edit mode by choosing 'attached to signals' from the 'Show annotations' menu in **wave**'s View window).

<down arrow>
Move the selected annotation down one signal (i.e., increment its **chan** field). This command works in multi-edit mode only.

keypad <5> (*<F2>*)
Insert an annotation at the current position of the pointer. (Click the middle button to do this using the mouse. Annotation editing must be enabled for this action to be successful.)

keypad <=> (*<F3>*)
Move the pointer toward the left.

keypad <>* (*<F4>*)
Move the pointer toward the right.

<Copy> (*<F6>*)
Copy the selected annotation to the Annotation Template.

<Find> (*<F9>*)
Search forward.

<ctrl><Find> (*<ctrl><F9>*)
Search backward.

<End> (*<shift><F9>*)
Advance to the end of the record.

<Home> (*<ctrl><shift><F9>*)
Move to the beginning of the record.

<PgDn> (*<F10>*)
Advance half a screen.

<ctrl><PgDn> (*<ctrl><F10>*)
Advance a full screen.

<PgUp> (*<shift><F10>*)
Move back half a screen.

<ctrl><PgUp> (*<ctrl><shift><F10>*)
Move back a full screen.

<Enter> (*<Return>*)
(Only if a link annotation has been selected.) Show the external data specified by the link using a Web browser; start the Web browser first if necessary.

BUGS

Under SunOS, once you have opened the Analyze window or have selected Print from the File menu, do not attempt to suspend **wave** (for example, by typing control-Z in the controlling terminal window). Under these circumstances, **wave** may exit immediately (without quit confirmation) and any unsaved edits may be lost. This problem is the result of a bug in the XView *termsw* package used for the Analysis Commands window. To avoid this bug, always run **wave** in the background under SunOS. The Linux, Mac OS X, MS Windows, and Solaris 2.x versions of the XView library do not have this bug.

If **wave** opens with an empty signal window, this may mean that the X server's backing store is disabled. **wave** versions 6.8 and later incorporate a workaround that avoids this problem. If you must use an earlier version of **wave**, enable backing store and restart the X server. (Using the X servers from the x.org or XFree86 projects, backing store can be enabled by inserting the line 'Option "backingstore"' in the 'Device' section(s) of the **xorg.conf** or **XF86Config-4** file. If your X server is normally started by a display manager such as **xdm**, close all windows and restart the server with *<ctrl><alt><backspace>*. Otherwise, log out, log in, and restart the X server manually if necessary.)

If this doesn't solve the problem, use any of **wave**'s navigation controls, or resize the signal window, to make the signals visible. On some 24-bit displays, this problem may be the result of an X server bug, and these methods will work around the problem. On some of these displays, text in the signal window may be invisible using overlay graphics mode; if this happens, use the **-S** option.

No more than one piped record (see the *WFDB Programmer's Guide*) can be viewed in a single invocation of **wave**. If the signal file is a pipe, it is possible only to search forward through it (although **wave** caches several of the most recently displayed windows, which can be reviewed in any case). Using the '>' button to move by half a frame does not work properly with piped input, nor does changing the display scales, since these actions require rereading the signals.

There appears to be a subtle incompatibility between XView-based applications such as **wave** and at least some X servers. The symptom of this problem is that **wave**'s View panel may be blank, and many warning messages from the notifier may appear in the controlling terminal window. This problem appears to occur only when all of the following are true: the X server is running on a multi-head display with Xinerama enabled, the user does not have root privileges, a **.Xdefaults** file exists, and **wave** or another XView application has run at least once since the X server was started.

A more serious incompatibility (which may be related to the subtle incompatibility noted above) appeared with the release in 2009 of the X.org version 1.6.3 X server, which freezes when any application that uses the XView library (such as **wave**) 'grabs' the mouse pointer. By default, XView applications do so in response to a left button click on any XView control. 'Grabs' can be disabled, and this behavior avoided, by using the **-Wfsdb** option available in **wave** and in other XView applications. In **wave** version 6.10 and later versions, the default behavior of XView has been changed to disable 'grabs', and this problem does not occur.

SEE ALSO

pschart(1), **xview**(7)
WAVE User's Guide (http://www.physionet.org/physiotools/wug/)
X Window System User's Guide: Open Look Edition (http://www.oreilly.com/openbook/openlook/)

AVAILABILITY

wave currently runs under FreeBSD, GNU/Linux, Mac OS X, MS-Windows with Cygwin/X, Solaris, and SunOS. It should be easily portable to any POSIX-compliant OS that can support X11 and XView. If you would like to use **wave** on a system other than those listed above, you will need to port XView to your system first (or purchase a commercial port if one is available). Sources for XView are available from PhysioNet (**www.physionet.org**, where the sources for **wave** itself are also available), **www.ibiblio.org**, and their mirrors. *We cannot offer assistance in porting XView; if you wish to try this, you are on your own.* If you successfully port the **cmdtool** terminal emulator application included in the XView sources, we will assist you in porting **wave** (this is much simpler than the XView port).

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/wave/

NAME

wfdb-config – print WFDB library version and configuration info

SYNOPSIS

wfdb-config [**--cflags**] [**--libs**] [**--version**]

DESCRIPTION

This program prints information about the WFDB library installation. Use it with one of these options:

--cflags Print options needed by **cc**(1) or **gcc**(1) to find the WFDB library's 'include' (*.h) files.

--libs Print options needed by **cc**(1), **gcc**(1), or **ld**(1) to find and link a program with the WFDB library (and, if **NETFILES** support is compiled into the WFDB library, with the *libwww* libraries).

--version
Print the version number of the most recent version of the WFDB library that has been installed.

Example

To compile *prog.c* with the WFDB library, use:

gcc ‘wfdb-config --cflags‘ prog.c ‘wfdb-config --libs‘

Additional options may be added to the command if needed (for example, to link to other libraries).

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wfdb-config.c

NAME

wfdb2mat – convert WFDB-compatible signal file to Matlab .mat file

SYNOPSIS

wfdb2mat -r *record* [*options ...*]

DESCRIPTION

This program converts the signals of any PhysioBank record (or one in any compatible format) into a *record***.mat** file that can be read directly using any version of Matlab, and a short *record***.hea** text file containing information about the signals (names, gains, baselines, units, sampling frequency, and start time/date if known). In addition, **wfdb2mat** writes a brief summary of this information to the standard output.

The **.mat** and **.hea** output files can also be read by any WFDB application as record *RECm*.

This program does not convert annotation files; for that task, **rdann**(1) is recommended.

The output **.mat** file contains a single matrix named **val**, containing raw (unshifted, unscaled) samples from the selected *record*. Using various options (below), one can select any time interval within a record, or any subset of the signals, which can be rearranged as desired within the rows of the matrix. Since **.mat** files are written in column-major order (i.e., all of column *n* precedes all of column *n+1*), each vector of samples is written as a column rather than as a row, so that the column number in the **.mat** file equals the sample number in the input record. If this seems odd, transpose your matrix after reading it!

This program writes version 4 MAT-file format output files (see http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/matfile_format.pdf for details). The samples are written as 32-bit signed integers in little-endian format if the record contains any format 24 or format 32 signals, as 8-bit unsigned integers if the record contains only 8-bit unsigned samples, or as 16-bit signed integers in little-endian format otherwise. Although version 5 and newer versions of Matlab normally use a different (less compact and more complex) format, they can read these files without difficulty. The advantage of version 4 MAT-file format, apart from compactness and portability, is that files in these formats are still WFDB-compatible, given the **.hea** file constructed by this program.

Options for **wfdb2mat** include:

-f *time* Begin at the specified *time*. By default, **wfdb2mat** starts at the beginning of the record.

-h Print a brief usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-l *interval*
Limit the amount of output to the specified time *interval* (in standard time format; default: no limit). If both **-l** and **-t** are used, **wfdb2mat** stops at the earlier of the two limits.

-s *signal-list*
Convert only the signals named in the *signal-list* (one or more input signal numbers or names, separated by spaces; default: print all signals). This option may be used to re-order or duplicate signals.

-S *signal*
Search for the first valid sample of the specified *signal* (a signal name or number) at or following the time specified with **-f** (or the beginning of the record if the **-f** option is not present), and begin converting at that time.

-t *time* Stop at the specified *time*. By default, **wfdb2mat** stops at the end of the record.

Example

To convert record **mitdb/200**, use this command:

wfdb2mat -r mitdb/200

This works even if the input files have not been downloaded; in this case, **wfdb2mat** reads them directly from the PhysioNet server.

The output files are **mitdb/200m.mat** and **mitdb/200m.hea**. Note that if a subdirectory of the current directory named **mitdb** did not exist already, it would be created by **wfdb2mat**. In addition, if the standard output of **wfdb2mat** has been saved in a file named **mitdb/200m.info**, then the converted data can be read and plotted in Matlab or Octave from within the **mitdb** directory by running the command:

plotATM('200m.mat', '200m.info')

(Download **http://physionet.org/physiotools/matlab/plotATM.m** and install it in your Matlab or Octave environment first.)

Note that when EDF (or EDF+, BDF, or BDF+) files are used as input, they may have empty 'physical dimension' (units) fields, which imply that the associated signals are dimensionless (for example, they may be event markers or categorical variables). In such cases, **wfdb2mat** records the signal units as **nd** (no dimension).

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

AVAILABILITY

This program is provided in the *convert* directory of the WFDB Software Package. Run **make** in that directory to compile and install it if it has not been installed already.

The PhysioNet ATM (http://physionet.org/cgi-bin/ATM) provides web access to **wfdb2mat** (select **Export signals as .mat** from the Toolbox).

SEE ALSO

a2m(1), **edf2mit**, **snip**(1), **wav2mit**(1), **xform**(1), **wfdb**(3), **header**(5)
http://physionet.org/physiotools/matlab/plotATM.m

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/convert/wfdb2mat.c

NAME

wfdbcat – copy WFDB files to standard output

SYNOPSIS

wfdbcat *file* [*file* ...]

DESCRIPTION

This program locates files anywhere in the WFDB path and copies them to the standard output. If linked with a version of the WFDB library that has been compiled with **NETFILES** support, **wfdbcat** can be particularly useful for retrieving files from remote web (HTTP) and FTP servers.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

setwfdb(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wfdbcat.c

NAME

wfdbcollate – collate WFDB records into a multi-segment record

SYNOPSIS

wfdbcollate -i *irec* [*irec ...*] **-o** *orec* [**-a** *annotator*]
wfdbcollate *orec first last* [**-a** *annotator*]
wfdbcollate -s *irec* **-o** *orec* [**-l** *segment-length*]

DESCRIPTION

A multi-segment record is the concatenation of one or more ordinary records. A multi-segment record is a ''virtual'' record, in the sense that it has no signal files of its own. Its header file contains a list of the records that comprise the multi-segment record. A multi-segment record may have associated annotation files, but these are independent of any annotation files that may exist for its constituent segments. It is permissible (though not particularly useful) to create a multi-segment record with only one segment; it is not permissible to use a multi-segment record as a segment within a multi-segment record, however.

wfdbcollate simply constructs an array of segment names, passing it to the WFDB library function *setmsheader* (see **wfdb**(3)) to create a multi-segment header file. In the first form of the command, *orec* is the name of the multi-segment (output) record to be created, and the *irec* arguments are the names of the (single-segment) input records that are to be included in the output record. At least one input record name must be specified.

In the second form of the command, *orec* is again the name of the multi-segment (output) record to be created, and *first* and *last* are numbers between 1 and 99999. In this case, *orec* must be 3 characters or fewer (longer names are truncated), and the names of the input records are derived by appending *first*, *first*+1, ..., *last* to *orec* (representing *first*, ..., as 5-digit zero-padded decimal numbers). Thus the command

wfdbcollate xyz 9 12

is equivalent to

wfdbcollate -o xyz -i xyz00009 xyz00010 xyz00011 xyz00012

Each segment must contain the same number of signals, and the sampling frequency must be the same for each segment. Each input record header must specify its record length (use **wfdbdesc**(1) to determine the input record length if necessary, then edit the input record header to include this information before using **wfdbcollate**). In most cases you will want to be sure that corresponding signals match in each segment, and that the gains, ADC zero levels, and numbers of samples per frame (see **header**(5)) also match. It is not necessary that the signal file formats match, however.

In the first two forms, **-a** *annotator* is optional; if included, it specifies the annotator name of annotation files associated with the input records, files to be concatenated to form a similarly-named annotation file for *orec*. Note that all of the files to be concatenated must have the same annotator name. It is not necessary that this annotator exist for each input record, however.

The third form of the command, which includes the **-s** option, can be used to split an existing record (*irec*) into multiple segments. In this mode, **wfdbcollate** first creates a set of segments from *irec*, then collates them into a multi-segment record. In this mode, the **-l** option may be used to specify a non-standard segment length, which must be no less than 15 seconds. By default, segments are 10 minutes long, although the last segment in the record may be shorter. The names of the segments created in this way are formed from the first three characters of *orec* and from a 5-digit zero-padded segment number, as in the second form of the command.

In most cases, multi-segment records are indistinguishable from single-segment records, from the point of view of applications built using the WFDB library (version 9.1 or later). Use **xform**(1) to generate a single-segment record from a multi-segment record if necessary (for example, to make it readable by an application built using an earlier version of the WFDB library). Note, however, that older applications can generally be updated without source changes simply by recompiling them and linking them with the current WFDB library.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

orec.hea	output header file
irec.hea	input header file(s)

BUGS

Under MS-DOS, this program is known as *wfdbcoll8*.

SEE ALSO

wfdbdesc(1), **xform**(1), **wfdb**(3), **header**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wfdbcollate.c

NAME

wfdbdesc – read signal specifications

SYNOPSIS

wfdbdesc *record* [**-readable**]

DESCRIPTION

This program reads specifications for the signals described in the *header* file for *record*. If the **-readable** option is present, **wfdbdesc** attempts to open the signal files, and it reports only on those that are readable.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

record.hea	header file
signal files	named in *record*.hea

AVAILABILITY

This program is provided in the *app* directory of the WFDB Software Package. Run **make** in that directory to compile and install it if it have not been installed already.

The PhysioNet ATM (http://physionet.org/cgi-bin/ATM) provides web access to **wfdbdesc** (select **Describe record** from the Toolbox).

SEE ALSO

setwfdb(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wfdbdesc.c

NAME

wfdbmap – make a synoptic map of a WFDB record

SYNOPSIS

wfdbmap -r *record* [**-a** *annotator* ...]
map-record -r *record* [**-a** *annotator* ...]

DESCRIPTION

wfdbmap generates a shell script containing commands and embedded data for creating a synoptic map of a WFDB-compatible *record*, optionally including one or more associated sets of annotations (specified by one or more *annotator* names following the **-a** option). When the script is run, it creates a PostScript-format 'map' of the WFDB record and its annotations, using **lwplt** from the **plt**(1) package. The maps displayed by the PhysioBank ATM are created in this way, with an additional conversion of the PostScript map to browser-compatible PNG format using **convert** from ImageMagick (http://www.imagemagick.org/).

map-record is a shell script that illustrates how **wfdbmap** and (indirectly) **plt** are used to create a map, and how to convert the PostScript map into a PNG-format map using **convert**.

For example, to make a map of mitdb/200 and its associated 'atr' annotations, run the command:

map-record mitdb/200 -a atr

The outputs of this command are **200.ps** and **200.png**. If other annotation files are available, their annotator names can be given as additional command-line arguments:

map-record *record* **-a** *ann1 ann2 ann3 ...*

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

plt(1), **setwfdb**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wfdbmap.c http://www.physionet.org/physiotools/wfdb/app/signal-colors.h http://www.physionet.org/physiotools/wfdb/app/map-record

NAME

wfdbtime – convert time to sample number, elapsed, and absolute time

SYNOPSIS

wfdbtime -r *record time* [*time* ...]

DESCRIPTION

Using the specified *record* as a reference for determining the length of a sample interval and the absolute time represented by sample number 0, this program accepts one or more *time* arguments (in WFDB standard time format) and produces one line on the standard output for each such argument. In each output line, the corresponding *time* is written as a sample number (in the form s*nnn*), as an elapsed time interval in hours, minutes, and seconds from the beginning of the *record* (in the form *hh:mm:ss.sss*), and as an absolute time and date (in the form [*hh:mm:ss.sss DD/MM/YYYY*]). If the base time for the record is undefined, the absolute time cannot be calculated, and in this case the elapsed time appears (a second time) instead.

Additional **-r** *record* arguments may be given in the same command, to reset the sample interval length and base time for any subsequent *time* arguments.

EXAMPLES

The command

```
wfdbtime -r mimicdb/237/237 0 10:0 s20 "[14:0:0 20/7/1995]" e
```

produces the output

```
s0          0:00.000      [12:40:38.000 20/07/1995]
s75000      10:00.000     [12:50:38.000 20/07/1995]
s20         0:00.160      [12:40:38.160 20/07/1995]
s595250     1:19:22.000   [14:00:00.000 20/07/1995]
s19199992   42:39:59.936  [07:20:37.936 22/07/1995]
```

Note that the input arguments need not be zero-padded, that an input argument in absolute time format must be quoted to protect the brackets from the shell, and that the input argument **e** is evaluated as the time of the last sample in the input record.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

FILES

record.hea	header file

SEE ALSO

time2sec(1)

http://www.physionet.org/physiotools/wpg/strtim.htm (for further details about standard time format and additional examples)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wfdbtime.c

NAME

wfdbwhich – find a WFDB file and print its pathname

SYNOPSIS

wfdbwhich [**-r** *record*] *filename*
wfdbwhich [**-r** *record*] *file-type record*

DESCRIPTION

This program searches the WFDB path (as specified by the environment variable **WFDB**, see **setwfdb**(1)) for a specified *filename*, or for a file of a specified *file-type* (e.g., 'hea' or 'atr') that belongs to a specified *record*. If the file can be found, its full pathname is written to the standard output, and **wfdbwhich** exits with an exit status of zero (indicating success). If the file cannot be found, a diagnostic message, including the current value of the WFDB path, is written to the standard error output, and *wfdbwhich* exits with a non-zero exit status.

If the WFDB path includes '%r', use the **-r** *record* option to specify the record name to be substituted for '%r'.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

setwfdb(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wfdbwhich.c

NAME

wqrs – single-channel QRS detector based on length transform

SYNOPSIS

wqrs -r *record* [*options ...*]

DESCRIPTION

wqrs attempts to locate QRS complexes in an ECG signal in the specified *record*. The detector algorithm is based on the length transform. The output of **wqrs** is an annotation file (with annotator name **wqrs**) in which all detected beats are labelled normal; the annotation file will also contain optional J-point annotations if the **-j** option (see below) is used.

wqrs can process records containing any number of signals, but it uses only one signal for QRS detection (signal 0 by default; this can be changed using the **-s** option, see below). **wqrs** is optimized for use with adult human ECGs. For other ECGs, it may be necessary to experiment with the sampling frequency as recorded in the input record's header file (see **header**(5)), the detector threshold (which can be set using the **-m** option), and the time constants indicated in the source file.

wqrs optionally uses the WFDB library's *setifreq* function to resample the input signal at 120 or 150 Hz (depending on the mains frequency, which can be specified using the **-p** option). **wqrs** performs well using input sampled at a range of rates up to 360 Hz and possibly higher rates, but it has been designed and tested to work best on signals sampled at 120 or 150 Hz.

Options include:

-d Dump the raw and length-transformed input samples in text format on the standard output, but do not detect or annotate QRS complexes.

-f *time* Begin at the specified *time* in *record* (default: the beginning of *record*).

-h Print a brief usage summary.

-H Read the signal files in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-j Find and annotate J-points (QRS ends) as well as QRS onsets.

-m *threshold*
Specify the detection *threshold* (default: 100 microvolts); use higher values to reduce false detections, or lower values to reduce the number of missed beats.

-p *frequency*
Specify the power line (mains) frequency used at the time of the recording, in Hz (default: 60). **wqrs** will apply a notch filter of the specified frequency to the input signal before length-transforming it.

-R Resample the input at 120 Hz if the power line frequency is 60 Hz, or at 150 Hz otherwise (default: do not resample).

-s *signal*
Specify the *signal* (number or name) to be used for QRS detection (default: 0).

-t *time* Process until the specified *time* in *record* (default: the end of the *record*).

-v Verbose mode: print information about the detector parameters.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

EXAMPLES

To mark QRS complexes in record 100 beginning 5 minutes from the start, ending 10 minutes and 35 seconds from the start, and using signal 1, use the command:

wqrs -r 100 -f 5:0 -t 10:35 -s 1

The output annotations may be read using (for example):

rdann -a wqrs -r 100

To evaluate the performance of this program, run it on the entire record, by:

wqrs -r 100

and then compare its output with the reference annotations by:

bxb -r 100 -a atr wqrs

SEE ALSO

bxb(1), **ecgpuwave**(1), **rdann**(1), **setwfdb**(1), **sqrs**(1)

Zong W, Moody GB, and Jiang D. A robust open-source algorithm to detect onset and duration of QRS complexes. *Computers in Cardiology* **30**:737–740 (2003).

AUTHORS

Wei Zong (wzong@mit.edu) and George B. Moody (george@mit.edu).

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wqrs.c

NAME

wrann – write a WFDB annotation file

SYNOPSIS

wrann -r *record* **-a** *annotator*

DESCRIPTION

wrann translates its standard input into an annotation file. The format of **wrann** input should be that produced by **rdann**(1) using its default settings. Specifically, the pipeline

rdann -r *record* **-a** *iann* **-f 0 | wrann -r** *record* **-a** *oann*

is guaranteed to produce an identical copy of the annotation file read by **rdann**, provided that the **aux** fields of the annotations do not contain embedded nulls.

The usual application for **wrann** is as an aid to annotation file editing: an annotation file may be translated into ASCII format using **rdann**, edited using a text editor, and then translated back into annotation file format using **wrann**.

Note the alternate format selected by **rdann**'s **-x** option is incompatible with **wrann**.

Versions of **wrann** included in version 10.4.20 (and later versions) of the WFDB Software Package set the time fields of output annotations to match the times in the first column of input if those times appear to be absolute times (beginning with '['). Otherwise, as in previous versions, **wrann** matches the sample numbers given in the second column of input. This capability can be used in a pipeline with **rdann** to copy a set of annotations between two records that overlap in time, even if their starting times or sampling frequencies are different.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

SEE ALSO

rdann(1), **setwfdb**(1)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wrann.c

NAME

wrsamp – write WFDB signal files

SYNOPSIS

wrsamp [*options ...*] [*column ...*]

DESCRIPTION

wrsamp reads text-format input and writes the specified *columns* in WFDB signal file format 16 (see **signal**(5)), either to the standard output or to a disk file (see the **-o** option below). If no *columns* are specified, all columns are written (but see the **-z** option below).

Normally, **wrsamp**'s input is line- and column-oriented, with *line separator characters* (usually ASCII linefeeds) separating input lines, and *field separator characters* (usually tabs, spaces, or commas) separating columns within each line. Columns need not be of constant width; the only requirement is that one or more field separator characters appear between adjacent columns. The output of **rdsamp**(1) is an example of an acceptable input format, as is CSV (comma-separated value) format.

If the first input line contains any alphabetic character, it is assumed to contain signal names (column headings), and these are copied to the output header file (see the **-o** option below). In this case, if the second input line also contains any alphabetic character, it is assumed to contain unit names (i.e., the names of the physical units of each signal), and these are also copied to the output header file. Spaces embedded within unit names are written as underscores in the header file.

Lines are identified by line number. The first line of input not containing any alphabetic character is line 0. Similarly, columns are identified by column number, and the leftmost column is column 0. Columns may be selected in any order, and any given column may be selected more than once, or omitted. The order of *column* arguments determines the order of the signals in the output (data from the first *column* specified are written as signal 0, etc.) If an entry in a specified column is "-" (i.e., flagged as missing or invalid), or if an entry in a specified column is any other non-numeric value, **wrsamp** records it as an invalid sample in its output.

If line 0 appears to begin with a timestamp (a field of the form *[hh:mm:ss.sss dd/mm/yyyy]*), **wrsamp** records it as the base time (starting time) in the output header file.

Options include:

-c Check that each input line contains the same number of fields. (This test is normally disabled, to allow for input files containing preambles, trailers, or occasional extra fields not intended to be read as samples.)

-d Dither the input before converting it to integer output, by adding a random value to each sample. The random values are selected from a triangular probability density function between -1 and +1. Dithering is appropriate whenever the output has a lower resolution than the input. Note that the RNG used to generate the pseudo-random values is started with a fixed seed, so that **wrsamp**'s output is strictly reproducible. Change the seed in the source and recompile to obtain a different realization of dither if desired.

-f *n* Start copying with line *n*. By default, **wrsamp** starts at the beginning of its standard input (line 0).

-F *n* Specify the sampling frequency (in samples per second per signal) for the output signals (default: 250). This option is useful only in conjunction with **-o**, since it affects the output header file only. This option has no effect on the output signal file, which contains one sample per signal for each line of input. If you wish to change the sampling frequency in the signal file, see **xform**(1).

-G *n* Specify the gain (in A/D units per millivolt) for the output signals (default: 200). To specify different gains for each output signal, provide a quoted list of values in place of *n* (see the examples below). This option is useful only in conjunction with **-o**, since it affects the output header file only. This option has no effect on the output signal file. If you wish to rescale samples in the signal file, use **-x**.

-h Print a usage summary.

-i *file* Read input from the specified *file* (default: standard input).

-l *n* Read up to *n* characters in each line (default: 1024). Longer lines are truncated (with a warning message identifying the line number of the offending line).

-o *record* Write the signal file in the current directory as *record***.dat**, and create a header file in the current directory for the specified *record*. By default, *wrsamp* writes the signal file to its standard output, and does not create a header file.

-O *format* Write the signal file in the specified *format* (default: 16). See **signal**(5) for descriptions and names of available formats.

-r *c* Interpret *c* as the input line separator (default: \n, the ASCII linefeed character). This option may be useful, for example, to read Macintosh files containing carriage-return delimited lines. Note that no special treatment is required for files containing both carriage returns and linefeeds.

-s *c* Interpret *c* as the input field separator (default: both spaces and tabs are treated as input field separators). If this option is used, *c* is the *only* character treated as a field separator.

-t *n* Stop copying at line *n* (line *n* is not processed). By default, **wrsamp** stops when it reaches the end of file on its standard input.

-x *n* Multiply all input samples by *n* (default: 1) before writing them to the output signal file. To specify different scaling factors for each signal, provide a quoted list of values in place of *n* (see the examples below).

-z Don't copy column 0 unless explicitly specified.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

Examples

rdsamp -r 100s | wrsamp -o 100w -F 360 1 2

This command creates a record named '100w' that is a copy of record '100s' (although the signal file format is different). If the **-F 360** option were omitted, the output signal file ('100w.dat') would be unchanged, but the header file for record '100w' would indicate that the sampling frequency was (the default) 250 Hz, rather than 360 Hz as in record 100s; this is because **wrsamp** has no other way of determining the sampling frequency of its input. Note that columns 1 and 2 of *wrsamp*'s input correspond to signals 0 and 1 respectively; column 0 is the sample number, not useful to **wrsamp**.

wrsamp -i in.txt -o out -G "100 100 50" -x "1 .5 -10 2" 4 1 0 3

This command creates a record named 'out' that contains signals derived from four columns of its input ('in.txt'). Notice that the argument of the -G (gain) option is the quoted string "100 100 50"; the effect is that the gains of the first two output signals are set to 100, and that of the third is 50. Since no explicit gain is specified for the fourth signal, it is assigned the same gain as the previous (third) signal (i.e., 50). Similarly, the quoted argument of the -x option specifies scaling factors applied to the samples before they are written to the output signal file: output signal 0 will be unscaled (scale factor 1), signal 1 will be halved (.5), signal 2 will be scaled by 10 and inverted (-10), and signal 3 will be doubled (2). Finally, note that the four columns selected from the input file have been rearranged, so that the leftmost column (0) will become output signal 2, etc.

SEE ALSO

rdsamp(1), **setwfdb**(1), **xform**(1), **signal**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/wrsamp.c

NAME

xform – sampling frequency, amplitude, and format conversion for WFDB records

SYNOPSIS

xform -i *input-record* [*options ...*]

DESCRIPTION

xform copies the signal files (and, optionally, annotation files) of the specified *input-record*. By default, all signals are copied in their entirety; using appropriate options, **xform** can be used to copy only a portion of the record, or only a subset of the signals, or both. *Options* are:

-a *annotator*
Copy the specified *annotator* as well as the signal files. Two or more *annotator* arguments, separated by spaces, can follow **-a**. An annotator supplied via the standard input may be specified using '-', but only immediately after **-a**; in this case only, annotations are copied to the standard output.

-c Clip the output (set any sample values that would fall outside of the range supported by the selected format to the maximum or minimum supported values). By default, the output is not clipped; rather, the values are wrapped around modulo the supported range (i.e., the excess high-order bits are simply discarded). Use of wrap-around can result in bizarre artifacts, but has the advantage that the affected portions of the output signals can (usually) be interpreted properly. Clipping mode is appropriate for testing algorithms or devices that must operate using a more restricted amplitude range than was used when digitizing the original record.

-d Dither the input by adding a pseudo-random value to each sample. The pseudo-random values are selected from a triangular probability density function between -1 and +1. Dithering is appropriate whenever the output has a lower resolution than the input, as may occur when changing the sampling frequency or gain. The **-d** option has no effect unless the sampling frequency or gain are changed in the output record. Note that the RNG used to generate the pseudo-random values is started with a fixed seed, so that **xform**'s output is strictly reproducible. Change the seed in the source and recompile to obtain a different realization of dither if desired.

-f *time* Begin at the specified *time* in the input record (default: the beginning of the record).

-h Print a usage summary.

-H Read the signals in high-resolution mode (default: standard mode). These modes are identical for ordinary records. For multifrequency records, the standard decimation of oversampled signals to the frame rate is suppressed in high-resolution mode (rather, all other signals are resampled at the highest sampling frequency).

-M Read the signals in multifrequency mode. Each signal (in a multifrequency record) is copied to the output record without changing its sampling frequency. In an ordinary record, this option has no effect other than to force the input and output sampling frequencies to be equal.

-n *new-record*
Create a *new-record* for the output signal files.

-N *new-record*
As above, but copy the signal descriptions from the header file for the record specified using the **-o** option (see below) rather than from the input record.

-o *output-record*
The header file for *output-record* (which must exist before running **xform**) determines the names, sampling frequency, formats (see **signal**(5)), gains, and ADC zero levels of the output signals. If the **-o** option is absent, **xform** prompts the user for the output specifications.

-s *signal-list*
Write only the signals named in the *signal-list* (one or more input signal numbers or names, separated by spaces; default: write all signals). This option may be used to re-order or duplicate signals.

-S *script*
Take answers to prompts from the specified *script* (a text file).

-t *time* Process until the specified *time* in the input record (default: continue to the end of the record).

-u Adjust annotation times as needed so that they are unique. If the output sampling frequency is less than that of the input, the times of closely-spaced annotations may coincide in the output, which may cause problems for some older WFDB applications. The **-u** option avoids this.

If a *new-record* is specified, a new header file is created after the signal file transformation is complete. The new header file, if created, contains the correct sample counts and checksums for the new signal files. Any output annotation files that are created as a result of using **–a** are associated with *new-record* if it has been specified, or with *output-record* otherwise. To process only a segment of the *input-record*, specify the starting and ending times using the **-f** and **-t** options.

Sampling frequency changes are performed by linear interpolation; any combination of input and output sampling frequencies is permissible. This interpolation method has the advantage of being reasonably fast, an important consideration since it is often necessary to operate on a million or more samples. Resampling noise is not a significant problem for the typical applications of **xform** (changing the sampling frequency by factors of five or less). Aliasing can be a problem, however, when the input sampling frequency is greater than the output sampling frequency. In such cases, if the input signals contain frequency components at or above half of the output sampling frequency, the input signals should be low-pass filtered (using, for example **fir**(1)) to remove these components before processing them with **xform**. Conversely, if the output sampling frequency is substantially greater than the input sampling frequency, resampling noise introduced at frequencies in excess of half of the input sampling frequency can be removed by low-pass filtering the output signals.

Normally, the ADC resolution fields in the header files are ignored, and scaling is determined by the ratios of the gain fields. An undefined (0) gain is considered equivalent to a gain of 200 ADC units per physical unit. An exception to this rule occurs if both input and output gains are undefined; in this case, scaling is determined by the difference in the ADC resolution fields, if any.

Also note that **xform** writes over any existing data files named in the header file for *output-record*; thus *output-record* should not be the name of an ordinary database record. Normally, the database signal files are read-only, and attempts to overwrite them are futile. For many applications the "piped records" *8* and *16* and the "local records" *8l* and *16l* will be found useful as output records.

If signal selection, scaling, and sampling frequency conversion are not needed, **snip**(1) is recommended as a faster alternative to **xform**.

ENVIRONMENT

It may be necessary to set and export the shell variable **WFDB** (see **setwfdb**(1)).

DIAGNOSTICS

As **xform** runs, it prints a '.' on the standard error output for each minute processed. If any of the output samples fall outside the range of values that can be properly represented using the specified output format, **xform** issues warnings but continues to process the record.

SEE ALSO

fir(1), **setwfdb**(1), **snip**(1), **signal**(5)

AUTHOR

George B. Moody (george@mit.edu)

SOURCE

http://www.physionet.org/physiotools/wfdb/app/xform.c

NAME

wfdb – Waveform Database library

SYNOPSIS

#include <wfdb/wfdb.h>

int adumuv(WFDB_Signal *s*, WFDB_Sample *adc_units*)
double aduphys(WFDB_Signal *s*, WFDB_Sample *adc_units*)
char *anndesc(int *annotation_code*)
int annopen(char **record*, WFDB_Anninfo **aiarray*, unsigned int *nann*)
char *annstr(int *annotation_code*)
int calopen(char **calibration_filename*)
char *datstr(WFDB_Date *d*)
char *ecgstr(int *annotation_code*)
int findsig(char **signal_name*)
void flushcal(void)
WFDB_Frequency getafreq(void)
int getann(WFDB_Annotator *a*, WFDB_Annotation **annotation*)
double getbasecount(void)
int getcal(char **description*, char **units*, WFDB_Calinfo **cal*)
WFDB_Frequency getcfreq(void)
int getgvmode(void)
WFDB_Frequency getiafreq(WFDB_Annotator *a*)
WFDB_Frequency getiaorigfreq(WFDB_Annotator *a*)
WFDB_Frequency getifreq(void)
int getseginfo(WFDB_Seginfo **segments)
char *getwfdb(void)
int getframe(WFDB_Sample **vector*)
char *getinfo(char **record*)
int getspf(void)
int getvec(WFDB_Sample **vector*)
void iannclose(WFDB_Annotator a)
int iannsettime(WFDB_Time *t*)
int isgsettime(WFDB_Group *signal_group*, WFDB_Time *t*)
int isigopen(char **record*, WFDB_Siginfo **siarray*, int *nsig*)
int isigsettime(WFDB_Time *t*)
char *mstimstr(WFDB_Time *t*)
WFDB_Sample muvadu(WFDB_Signal *s*, int *microvolts*)
int newcal(char **calibration_filename*)
int newheader(char **record*)
void oannclose(WFDB_Annotator a)
int osigfopen(WFDB_Siginfo **siarray*, unsigned int *nsig*)
int osigopen(char **record*, WFDB_Siginfo **siarray*, unsigned int *nsig*)
WFDB_Sample physadu(WFDB_Signal *s*, double *v*)
int putann(WFDB_Annotator *a*, WFDB_Annotation **annotation*)
int putcal(WFDB_Calinfo **cal*)
int putinfo(char **info*)
int putvec(WFDB_Sample **vector*)
WFDB_Frequency sampfreq(char **record*)
WFDB_Sample sample(WFDB_Signal *s*, WFDB_Time *t*)
int sample_valid(void)
void setafreq(WFDB_Frequency *annotation_clock_frequency*)
int setanndesc(int *annotation_code*, char **annotation_description*)
int setannstr(int *annotation_code*, char **annotation_mnemonic_string*)
void setbasecount(double *base_count*)
int setbasetime(char **time_string*)

void setcfreq(WFDB_Frequency *counter_frequency*)
void setiafreq(WFDB_Annotator *a*, WFDB_Frequency *annotation_clock_frequency*)
int setifreq(WFDB_Frequency *getvec_frequency*)
void setwfdb(char **database_path_string*)
int setecgstr(int *annotation_code*, char **annotation_mnemonic_string*)
void setgvmode(int *mode*)
int setheader(char **record*, WFDB_Siginfo **siarray*, unsigned int *nsig*)
int setibsize(int *size*)
int setmsheader(char **record*, char ***seg_names*, unsigned int *nsegments*)
int setobsize(int *size*)
int setsampfreq(WFDB_Frequency *sampling_frequency*)
int strann(char **annotation_mnemonic_string*)
WFDB_Date strdat(char **date_string*)
int strecg(char **annotation_mnemonic_string*)
WFDB_Time strtim(char **time_string*)
char *timstr(WFDB_Time *t*)
WFDB_Time tnextvec(WFDB_Signal *s*, WFDB_Time *t*)
int ungetann(WFDB_Annotator *a*, WFDB_Annotation **annotation*)
const char *wfdbcflags(void)
const char *wfdbdefwfdb(void)
const char *wfdbdefwfdbcal(void)
char *wfdberror(void)
char *wfdbfile(char **type*, char **record*)
void wfdbflush(void)
int wfdbgetskew(WFDB_Signal *s*)
long wfdbgetstart(WFDB_Signal *s*)
int wfdbinit(char **record*, WFDB_Anninfo **aiarray*, unsigned int *nann*,
WFDB_Siginfo **siarray*, unsigned int *nsig*)
const char *wfdbldflags(void)
void wfdbmemerr(int exit_on_error)
int wfdbputprolog(char **prolog*, long *bytes*, WFDB_Signal *s*)
void wfdbquiet(void)
void wfdbquit(void)
void wfdbsetskew(WFDB_Signal *s*, int *skew*)
void wfdbsetstart(WFDB_Signal *s*, long *byte_offset*)
void wfdbverbose(void)
const char *wfdbversion(void)

DESCRIPTION

Waveform databases (including the MIT-BIH Arrhythmia Database, the AHA Database for Evaluation of Ventricular Arrhythmia Detectors, and the European ST-T Database) are accessible to applications written in C and C++ via the functions defined in the WFDB library. Under UNIX, programs may be linked with the WFDB library by using the **-lwfdb** option at the end of the C or C++ compiler command. The functions are described in detail in the reference below.

FILES

UNIX systems:

/usr/lib/libwfdb.a	standard (statically bound) library
/usr/lib/libwfdb.so	
/usr/lib/libwfdb.so.*M.N*	shareable library (bound at run-time, not available on all systems). On some systems, one of these pathnames is a link to the other, and both are needed; on others, only one of the pathnames is needed.
/usr/lib/libwfdb.sa	stubs for linking with applications that use **libwfdb.so** (not needed on all systems).

The location of these files may vary on some systems.

MS-DOS/MS Windows systems:

wfdb.lib
: standard (small memory model) library

wfdbl.lib
: large memory model library

wfdb.dll
: dynamic link library for MS Windows

wfdbdll.lib
: stubs for linking with applications that use **wfdb.dll**

SEE ALSO

WFDB Programmer's Guide
: On systems that support GNU emacs, the *Guide* may be available on-line using emacs *info*; from within *emacs*, type control-H followed by *i* to find out. An HTML version may be installed on your system (in */usr/help/html/wpg*); the most recent version can be viewed on-line at **http://www.physionet.org/physiotools/wpg/**.

The WFDB library can also be used with Fortran programs; see *wfdbf*(3) and the *Guide* for details.

DIAGNOSTICS

All functions that return an **int** indicate errors with negative values. Depending on context, zero returns may indicate success or failure. Positive values indicate success. Most errors other than EOF are accompanied by diagnostics on the standard error output.

AUTHORS

George B. Moody (george@mit.edu), with contributions from many sources. The predecessor of the WFDB library was originally implemented in C by George Moody and Ted Baker, based on earlier designs by Paul Schluter and Larry Siegal. Other contributors of code and ideas include Paul Albrecht, Mike Dakin, Phil Devlin, Scott Greenwald, David Israel, Roger Mark, Joe Mietus, and Warren Muldrow. Pat Hamilton and Bob Farrell contributed ports, to MacOS and Win32 respectively.

SOURCES

http://www.physionet.org/physiotools/wfdb/lib/

NAME

wfdbf – Waveform Database library wrappers for Fortran

SYNOPSIS

```
implicit integer(a-z)
real aduphys, getbasecount, getcfreq, getifreq, sampfreq, getafreq, getiafreq, getiaorigfreq
character aux(256), desc(80), filetype(32), fname(40), name(20), pathname(80), record(16), string(32), units(20), prolog(1000), version(80), options(80)
integer a, adcres, adczero, ampl, anntyp, baseline, bsize, caltype, chan, cksum, date, dummy, fmt, group, initval, microvolts, mode, nann, nsamp, nsig, num, s, spf, stat, subtyp, time, v(32), value, bytes
real gain, frequency, high, low, scale

setanninfo(a, name, stat)
getsiginfo(s, fname, desc, units, gain, initval, group, fmt, spf, bsize, adcres, adczero, baseline, nsamp, cksum)
setsiginfo(s, fname, desc, units, gain, initval, group, fmt, spf, bsize, adcres, adczero, baseline, nsamp, cksum)
annopen(record, nann)
isigopen(record, nsig)
osigopen(record, nsig)
osigfopen(nsig)
wfdbinit(record, nann, nsig)
findsig(desc)
setgvmode(mode)
getgvmode(dummy)
getspf(dummy)
getvec(v)
getframe(v)
putvec(v)
getann(a, time, anntyp, subtyp, chan, num, aux)
ungetann(a, time, anntyp, subtyp, chan, num, aux)
putann(a, time, anntyp, subtyp, chan, num, aux)
isigsettime(time)
isgsettime(group, time)
tnextvec(s, time)
iannsettime(time)
ecgstr(code, string)
strecg(string)
setecgstr(code, string)
annstr(code, string)
strann(string)
setannstr(code, string)
anndesc(code, string)
setanndesc(code, string)
setafreq(frequency)
getafreq(dummy)
setiafreq(a, frequency)
getiafreq(a)
getiaorigfreq(a)
iannclose(a)
oannclose(a)
timstr(time, string)
mstimstr(time, string)
strtim(string)
datstr(date, string)
strdat(string)
```

adumuv(s, ampl)
muvadu(s, microvolts)
aduphys(s, ampl)
physadu(s, value)
sample(s, time)
sample_valid(dummy)
calopen(fname)
getcal(desc, units, low, high, scale, caltype)
putcal(desc, units, low, high, scale, caltype)
newcal(fname)
flushcal(dummy)
getinfo(record, string)
putinfo(string)
setinfo(record)
wfdb_freeinfo(dummy)
newheader(record)
setheader(record, nsig)
wfdbgetskew(s)
wfdbsetiskew(s, value)
wfdbsetskew(s, value)
wfdbgetstart(s)
wfdbsetstart(s, value)
wfdbputprolog(prolog, bytes, s)
wfdbquit(dummy)
sampfreq(record)
setsampfreq(frequency)
getcfreq(dummy)
setcfreq(frequency)
getifreq(dummy)
setifreq(frequency)
getbasecount(dummy)
setbasecount(frequency)
setbasetime(string)
wfdbquiet(dummy)
wfdbverbose(dummy)
wfdberror(string)
setwfdb(string)
getwfdb(string)
resetwfdb(dummy)
setibsize(value)
setobsize(value)
wfdbfile(filetype, record, pathname)
wfdbflush(dummy)
wfdbmemerr(mode)
wfdbversion(version)
wfdbldflags(options)
wfdbcflags(options)
wfdbdefwfdb(string)
wfdbdefwfdbcal(fname)
isann(anntyp)
isqrs(anntyp)
setisqrs(anntyp, value)
map1(anntyp)
setmap1(anntyp, value)

map2(anntyp)
setmap2(anntyp, value)
ammap(anntyp)
mamap(anntyp, subtyp)
annpos(anntyp)
setannpos(anntyp, value)

DESCRIPTION

Fortran programs can use the WFDB library to read and write waveform database files. Differences in argument-passing conventions between Fortran and C (the language of the WFDB library) require the use of a set of wrappers as an interface between the library and Fortran code that invokes its functions. These wrappers are contained within 'wfdbf.c', provided in the 'fortran' directory of the WFDB software package. When the WFDB Software Package is installed, a copy of 'wfdbf.c' is placed in the same directory as 'wfdb.h' (normally, /usr/include/wfdb).

Most of these wrapper subroutines behave like their similarly-named counterparts in the WFDB library. The functions setanninfo, setsiginfo, and getsiginfo do not have direct equivalents in the WFDB library; they are provided in order to permit Fortran programs to read and write data structures passed to and from several of the WFDB library functions. Since the contents of these structures are directly accessible by C programs, these functions are not needed in the C library.

Before using annopen, set up the annotation information structures using setanninfo. After using isigopen or osigopen, use getsiginfo to obtain the contents of the signal information structures if necessary. Before using osigfopen or setheader, use setsiginfo to set the contents of the signal information structures. Before using wfdbinit, use setanninfo and setsiginfo to set the contents of the annotation and signal information structures.

To use these wrappers, call them as shown above, then compile your code together with wfdbf.c and link to the WFDB library. If you are using the GNU g77 compiler (recommended), do so using a command such as:

g77 -o foo foo.f -DFIXSTRINGS /usr/include/wfdb/wfdbf.c -lwfdb

The wrappers include optionally compiled code that converts traditional space-terminated Fortran strings to null-terminated C strings and vice versa. This code is compiled if the symbol FIXSTRINGS is defined, as in the g77 command above. If you use a different Fortran compiler, this code may not be necessary. See 'fortran/README' for further information about using the WFDB Fortran wrappers.

SEE ALSO

WFDB Programmer's Guide

On systems that support GNU emacs, the *Guide* may be available on-line using emacs *info*; from within **emacs**, type control-H followed by *i* to find out. An HTML version may be installed on your system (in */usr/help/html/wpg*); the most recent version can be viewed on-line at **http://www.physionet.org/physiotools/wpg/**.

AUTHOR

George B. Moody (george@mit.edu)

SOURCES

http://www.physionet.org/physiotools/wfdb/fortran/wfdbf.c

NAME

annot – WFDB annotation file formats

SYNOPSIS

#include <wfdb/ecgcodes.h>

DESCRIPTION

Programs compiled with the WFDB library (**-lwfdb**) can read annotation files in two formats. The preferred format (MIT format) is compact (averaging slightly over two bytes per annotation) and extensible, and is normally used for on-line annotation files. The alternative format (AHA DB distribution format) uses 16 bytes per annotation, and is normally used only for exchange of files between institutions on 9-track tape. Both formats are binary, but readable on any machine without reformatting. WFDB library applications can distinguish between the formats automatically when opening a file for input.

MIT format:

Each annotation occupies an even number of bytes. The first byte in each pair is the least significant byte. The six most significant bits (*A*) of each byte pair are the annotation type code, and the ten remaining bits (*I*) specify the time of the annotation, measured in sample intervals from the previous annotation (or from the beginning of the record for the first annotation). If 0 < *A* <= **ACMAX**, then *A* is defined in **<wfdb/ecgcodes.h>**. Several other possibilities exist:

A = **SKIP** [59.]
: *I* = 0; the next four bytes are the interval in PDP-11 long integer format (the high 16 bits first, then the low 16 bits, with the low byte first in each pair).

A = **NUM** [60.]
: *I* = annotation **num** field for current and subsequent annotations; otherwise, assume previous annotation **num** (initially 0).

A = **SUB** [61.]
: *I* = annotation **subtyp** field for current annotation only; otherwise, assume **subtyp** = 0.

A = **CHN** [62.]
: *I* = annotation **chan** field for current and subsequent annotations; otherwise, assume previous **chan** (initially 0).

A = **AUX** [63.]
: *I* = number of bytes of auxiliary information (which is contained in the next *I* bytes); an extra null, not included in the byte count, is appended if *I* is odd.

A = *I* = 0: End of file.

AHA format:

All annotations occupy exactly 16 bytes. Within each block, the first byte is unused, the second byte contains the AHA annotation code (an ASCII character; see **<wfdb/ecgmap.h>**), the third through sixth bytes contain the time (see below) in PDP-11 long integer format as above, and the seventh and eighth bytes contain an annotation serial number.

In annotation files taken directly from the AHA database distribution tapes, the last eight bytes in each annotation are unused, and the time is given in milliseconds measured from the beginning of the annotated segment of the record. In AHA-format annotation files generated by WFDB library applications, annotation times are given in sample intervals from the beginning of the record, and the last eight bytes of each annotation contain the MIT annotation subtype (in the ninth byte), the MIT annotation code (in the tenth byte), and up to six ASCII characters (in the remaining bytes) used to describe **RHYTHM** and **NOTE** annotations.

SEE ALSO

header(5), **signal**(5), **wfdbcal**(5)
WFDB Programmer's Guide

AUTHOR

George B. Moody (george@mit.edu). The original MIT annotation format was designed by Paul Schluter, and the AHA annotation format was designed by Russ Hermes.

NAME

header – WFDB header file format

DESCRIPTION

For each database record, a header file specifies the names of the associated signal files and their attributes. Programs compiled with the WFDB library (*-lwfdb*) can read header files created by *newheader* (see **wfdb**(3)). Header files contain line- and field-oriented ASCII text. ASCII linefeed characters separate lines (which may not contain more than 255 characters each, including the linefeed), and spaces or tabs separate fields (except as noted below). Beginning with WFDB library version 6.1, an ASCII carriage return character may precede each linefeed. Fields not specifically designated below as optional must be present.

Header files contain at a minimum a *record line*, which specifies the record name, the number of segments, and the number of signals. Header files for ordinary records (those that contain one segment) also contain a *signal specification line* for each signal. Header files for multi-segment records (supported by WFDB library version 9.1 and later versions) contain a *segment specification line* for each segment; see the section on multi-segment records below for details.

Comment lines may appear anywhere in a header file. The first printing character in a comment line must be '#'. Comment lines that follow the last signal specification line are treated specially (see *Info strings*, below). All other comment lines are ignored by WFDB library functions that read header files.

Record line

The first non-empty, non-comment line is the *record line*. It contains information applicable to all signals in the record. Its fields are, from left to right:

record name

A string of characters that identify the record. The record name may include letters, digits and underscores ('_') only.

number of segments [optional]

This field, if present, is *not* separated by whitespace from the record name field; rather, it follows a '/', which serves as a field separator. If the field is present, it indicates that the record is a multi-segment record containing the specified number of segments, and that the header file contains segment specification lines rather than signal specification lines. The number of segments must be greater than zero. A value of 1 in this field is legal, though unlikely to be useful.

number of signals

Note that this is not necessarily equal to the number of signal files, since two or more signals can share a signal file. This number must not be negative; a value of zero is legal, however.

sampling frequency (in samples per second per signal) [optional]

This number can be expressed in any format legal for **scanf**(3) input of floating point numbers (thus '360', '360.', '360.0', and '3.6e2' are all legal and equivalent). The sampling frequency must be greater than zero; if it is missing, a value of 250 (**DEFREQ**, defined in *<wfdb/wfdb.h>*) is assumed.

counter frequency (in ticks per second) [optional]

This field (a floating-point number, in the same format as the sampling frequency) can be present only if the sampling frequency is also present. It is *not* separated by whitespace from the sampling frequency field; rather, it follows a '/', which serves as a field separator. The sampling and counter frequencies are used by *strtim* to convert strings beginning with 'c' into sample intervals. Typically, the counter frequency may be derived from an analog tape counter, or from page numbers in a chart recording. If the counter frequency is absent or not positive, it is assumed to be equal to the sampling frequency. WFDB library versions 5.1 and earlier ignore the counter frequency field.

base counter value [optional]

This field can be present only if the counter frequency is also present. It is *not* separated by whitespace from the counter frequency field; rather, it is surrounded by parentheses, which delimit it. The base counter value is a floating-point number that specifies the counter value corresponding to

sample 0. If absent, the base counter value is taken to be zero. WFDB library versions 5.1 and earlier ignore the base counter value field.

number of samples per signal [optional]
This field can be present only if the sampling frequency is also present. If it is zero or missing, the number of samples is unspecified and checksum verification of the signals is disabled.

base time [optional]
This field can be present only if the number of samples is also present. It gives the time of day that corresponds to the beginning of the record, in HH:MM:SS format (using a 24-hour clock; thus 13:05:00, or 13:5:0, represent 1:05 pm). If this field is absent, the time-conversion functions assume a value of 0:0:0, corresponding to midnight.

base date [optional]
This field can be present only if the base time is also present. It contains the date that corresponds to the beginning of the record, in DD/MM/YYYY format (e.g., 25/4/1989 is 25 April 1989).

Signal specification lines

Each non-empty, non-comment line following the record line in a single-segment record contains specifications for one signal, beginning with signal 0. Header files must contain valid signal specification lines for at least as many signals as were indicated in the record line (the first non-empty, non-comment line in the file). Any extra signal specification lines are not read by WFDB library functions. From left to right in each line, the fields are:

file name
The name of the file in which samples of the signal are kept. The environment variable **WFDB** (the database path) lists the directories in which signal files (as well as WFDB header and annotation files) are found; normally **WFDB** should include an initial empty component, so that signal files can be kept in any directory if they are designated by absolute path names in the header file. If the file name specifies that the signal file is to be found in a directory that is not already in **WFDB**, that directory is appended to the end of **WFDB** (by functions that read header files in WFDB library version 6.2 and later versions). Although the record name is usually part of the signal file name, this convention is not a requirement (see, e.g., examples 3, 4, and 5 below). Note that several signals can share the same file (i.e., they can belong to the same signal group); all entries for signals that share a given file must be consecutive, however. The file name '-' refers to the standard input or output. The sum of the lengths of the file name and description fields (see below) is limited to 80 characters.

format This field is an integer that specifies the storage format of the signal. All signals in a given group are stored in the same format. The most common formats are format 8 (eight-bit first differences) and format 16 (sixteen-bit amplitudes); see **signal**(5) (or *<wfdb/wfdb.h>*) for a list of other supported formats. The following three optional fields, if present, are bound to the format field (i.e., not separated from it by whitespace); they may be considered as format modifiers, since they further describe the encoding of samples within the signal file.

samples per frame [optional]
If present, this field follows an 'x' that serves as a field separator. Normally, all signals in a given record are sampled at the (base) sampling frequency as specified in the record line; in this case, the number of samples per frame is 1 for all signals, and this field is conventionally omitted. If the signal was sampled at some integer multiple, *n*, of the base sampling frequency, however, each frame (set of samples returned by *getframe*) contains *n* samples of the signal, and the value specified in this field is also *n*. (Note that non-integer multiples of the base sampling frequency are not supported.) WFDB library versions 8.3 and earlier ignore this field if it is present, and cannot properly read signal files that contain more than one sample per signal per frame.

skew [optional]
If present, this field follows a ':' that serves as a field separator. Ideally, within a given record, samples of different signals with the same sample number are simultaneous (within one sampling interval). If this is not the case (as, for example, when a multitrack analog tape recording is

digitized and the azimuth of the playback head does not match that of the recording head), the skew between signals can sometimes determined (for example, by locating recorded waveform features with known time relationships, such as calibration signals). If this has been done, the skew field may be inserted into the header file to indicate the (positive) number of samples of the signal that are considered to *precede* sample 0. These samples, if any, are included in the checksum, but cannot be returned by *getvec* or *getframe* (thus the checksum need not be changed if the skew field is inserted or modified). WFDB library versions 9.1 and earlier ignore this field if it is present; later versions correctly deskew signals in accordance with the contents of this field.

byte offset [optional]

If present, this field follows a '+' that serves as a field separator. Normally, signal files include only sample data. If a signal file includes a preamble, however, this field specifies the offset in bytes from the beginning of the signal file to sample 0 (i.e., the length of the preamble). Data within the preamble is not included in the signal checksum. Note that the byte offset must be the same for all signals within a given group (use the skew field to correct for intersignal skew). This feature is provided only to simplify the task of reading signal files not generated using the WFDB library; the WFDB library does not support any means of writing such files, and byte offsets must be inserted into header files manually. WFDB library versions 4.4 and earlier ignore byte offsets; these versions return any preamble data as samples.

ADC gain (ADC units per physical unit) [optional]

This field is a floating-point number that specifies the difference in sample values that would be observed if a step of one physical unit occurred in the original analog signal. For ECGs, the gain is usually roughly equal to the R-wave amplitude in a lead that is roughly parallel to the mean cardiac electrical axis. If the gain is zero or missing, this indicates that the signal amplitude is uncalibrated; in such cases, a value of 200 (**DEFGAIN**, defined in *<wfdb/wfdb.h>*) ADC units per physical unit may be assumed.

baseline (ADC units) [optional]

This field can be present only if the ADC gain is also present. It is *not* separated by whitespace from the ADC gain field; rather, it is surrounded by parentheses, which delimit it. The baseline is an integer that specifies the sample value corresponding to 0 physical units. If absent, the baseline is taken to be equal to the ADC zero. Note that the baseline need not be a value within the ADC range; for example, if the ADC input range corresponds to 200–300 degrees Kelvin, the baseline is the (extended precision) value that would map to 0 degrees Kelvin. WFDB library versions 5.0 and earlier ignore baseline fields.

units [optional]

This field can be present only if the ADC gain is also present. It follows the baseline field if that field is present, or the gain field if the baseline field is absent. It is not separated by whitespace from the previous field; rather, it follows a '/', which serves as a field separator. The units field is a character string without embedded whitespace that specifies the type of physical unit. If the units field is absent, the physical unit may be assumed to be one millivolt. WFDB library versions 4.7 and earlier ignore units fields.

ADC resolution (bits) [optional]

This field can be present only if the ADC gain is also present. It specifies the resolution of the analog-to-digital converter used to digitize the signal. Typical ADCs have resolutions between 8 and 16 bits. If this field is missing or zero, the default value is 12 bits for amplitude-format signals, or 10 bits for difference-format signals (unless a lower value is specified by the *format* field).

ADC zero [optional]

This field can be present only if the ADC resolution is also present. It is an integer that represents the amplitude (sample value) that would be observed if the analog signal present at the ADC inputs had a level that fell exactly in the middle of the input range of the ADC. For a bipolar ADC, this value is usually zero, but a unipolar (offset binary) ADC usually produces a non-zero value in the middle of its range. Together with the ADC resolution, the contents of this field can be used to determine the range of possible sample values. If this field is missing, a value of zero is

assumed.

initial value [optional]
This field can be present only if the ADC zero is also present. It specifies the value of sample 0 in the signal, but is used only if the signal is stored in difference format. If this field is missing, a value equal to the ADC zero is assumed.

checksum [optional]
This field can be present only if the initial value is also present. It is a 16-bit signed checksum of all *samples* in the signal. (Thus the checksum is independent of the storage format.) If the entire record is read without skipping samples, and the header's record line specifies the correct number of samples per signal, this field is compared against a computed checksum to verify that the signal file has not been corrupted. A value of zero may be used as a field placeholder if the number of samples is unspecified.

block size [optional]
This field can be present only if the checksum is present. This field is an integer and is usually zero. If the signal is stored in a file that must be read in blocks of a specific size, however, this field specifies the block size in bytes. (On UNIX systems, this is the case only for character special files, corresponding to certain tape and raw disk files. If necessary, the block size may be given as a negative number to indicate that the associated file lacks I/O driver support for **fseek**(3) operations.) All signals belonging to the same signal group have the same block size.

description [optional]
This field can be present only if the block size is present. Any text between the block size field and the end of the line is taken to be a description of the signal. When creating new records, follow the style used to document the signals in existing header files. Unlike the other fields in the header file, the description may include embedded spaces; note that whitespace between the block size and description fields is not considered to be part of the description, however. If the description is missing, the WFDB library functions that read header files supply a description of the form 'record *rec*, signal *n*' (shortened to 'signal *n*' by many WFDB applications).

Info strings

Comment lines that follow the last signal specification line in a header file can be read and written by the WFDB library functions *getinfo* and *putinfo*; the contents of these lines (excluding the initial '#' comment character) are referred to as 'info strings'. There must be no whitespace preceding the initial '#' in any line that is to be recognized by *getinfo*.

Multi-segment records

Each non-empty, non-comment line following the record line in the top-level header file of a multi-segment record contains specifications for one segment, beginning with segment 0. (Info strings cannot be used in the top-level header file of a multi-segment record.) Top-level header files must contain valid segment specification lines for at least as many segments as were indicated in the record line. Any extra segment specification lines are not read by WFDB library functions.

A *segment* is simply an ordinary (single-segment) record, with its own header and signal files. By including segments in a multi-segment record, the signals within them can be read by WFDB applications as if they were continuous signals, beginning with those in segment 0 and continuing with those in segment 1, with no need for the applications to do anything special to move from one segment to another. The only restrictions are that segments cannot themselves contain other segments (they *must* be single-segment records), the sampling frequencies must not change from segment to segment, and the number of samples per signal must be defined for each segment in the record line of the segment's own header file.

Two types of multi-segment records are defined. In a *fixed-layout* record, the arrangement of signals is constant across all segments, and the signal gain, baseline, units, ADC resolution and zero, and description match for corresponding signals in all segments (these recommendations are not enforced by the WFDB library, but existing applications are likely to behave unpredictably if they are not followed). Note, however, that it is not necessary to use the same signal storage format in all segments, and significant space savings may be possible in some cases by selecting an optimal format for each segment. Each segment of a

fixed-layout record is an ordinary record containing one or more samples.

In a *variable-layout* record, the arrangement of signals may vary, signals may be absent in some segments, and the gains and baselines may change between segments. A variable-layout record can be identified by the presence of a *layout segment*, which must be segment 0 and must have a length of 0 samples. The layout segment has no associated signal files; its header file specifies the desired arrangement of signals and their gains and baselines. Signal file names in a layout segment header are recorded as '~'. When read using WFDB library version 10.3.17 or later, the signals of a variable layout record are rearranged, shifted, and rescaled as needed in order to present the signals in the arrangement and with the gains and baselines specified in the layout segment header.

Segment specification lines

Each segment specification line contains the following fields, separated by whitespace:

record name

A string of characters identifying the single-segment record that comprises the segment. As in the record line, the record name may include letters, digits, and underscores ('_') only.

number of samples per signal

This number must match the number specified in the header file for the single-segment record that comprises the segment.

Variable-layout records may contain *null segments*, which can be identified if the record name given in the segment specification line is '~'. The number of samples per signal indicates the length of the null segment; when read, these samples have the value WFDB_INVALID_DATA (defined in <wfdb/wfdb.h>). Null segments do not have associated header or signal files.

Examples:

Example 1 (MIT DB record 100):

```
100 2 360 650000 0:0:0 0/0/0
100.dat 212 200 11 1024 995 -22131 0 MLII
100.dat 212 200 11 1024 1011 20052 0 V5

# 69 M 1085 1629 x1
# Aldomet, Inderal
```

This header specifies 2 signals each sampled at 360 Hz, each 650000 samples (slightly over 30 minutes) long. The starting time and date were not recorded; in the example, the defaults are shown, but they might be omitted without changing the meaning of the header file. Each signal is stored in 12-bit bit-packed format (2 samples per 3 bytes; see **signal**(5) for details), and one file contains both signals. Since the filename given (*100.dat*) does not include path information, WFDB library-based programs will find the signal file only if it is located in one of the directories specified by the **WFDB** environment variable. The gain for each signal was the (default) 200 ADC units per millivolt (the default physical unit), and the ADC had 11-bit resolution and an offset such that its output was 1024 ADC units given an input exactly in the middle of its range. The baseline is not given explicitly, but may be assumed to be equal to the ADC zero value of 1024. The first samples acquired had values of 995 and 1011 (i.e., both signals began slightly below 0 VDC). The checksums of the 650000 samples are -22131 and 20052, and I/O may be performed in blocks of any desired size (since the block size fields are zero). The signal descriptions specify which leads were used (MLII: modified lead II). Finally, the last two lines contain 'info strings'. (In this example, the first info string specifies the sex and age of the subject and data about the recording, and the second lists the subject's medications. The contents and format of info strings vary between databases; it is not wise to rely on the presence of specific data in info strings, since their use in header files is optional.)

Example 2 (AHA DB record 7001):

```
7001 2 250 525000
/db1/data0/d0.7001 8 100 10 0 -53 -1279 0 ECG signal 0
/db1/data1/d1.7001 8 100 10 0 -69 15626 0 ECG signal 1
```

This header illustrates how on-line AHA DB records were formerly kept at MIT. Note that the sampling frequency and ADC specifications differ from the previous example. In this example, each signal is kept in

its own signal file, specified by its absolute pathname. As shown here, AHA DB records may be kept in 8-bit first difference format, but the sampling rate requires that the signals be scaled down (from 12-bit to 10-bit ADC resolution) to stay within the slew rate limits imposed by the format. Note that signal checksums (-1279 and 15626 in this example) are derived from the reconstructed sample values, and not from the first differences; thus they should not change if the signals are reformatted.

Example 3 (Local record 8l):

```
8l 16
data0 8
data1 8
 ...
data15 8
```

This example illustrates how relative pathnames can be used for user-created records. If *data** files in the proper format are created in any of the directories named by the **WFDB** environment variable, they become the signal files for record 8l.

Example 4 (Piped record 16x4):

```
# Piped record 16x4.  Use this record to read or write 4 signals
# using the standard I/O.
16x4 4
- 16
- 16
- 16
- 16
```

This example illustrates several features not seen in the earlier examples. The special file name '-' means that samples will be read from the standard input or written to the standard output when using this record. All four signals are associated with the same file. The signals are kept in 16-bit amplitude format. The example includes two comment lines, which are ignored by the WFDB library functions that read header files.

Example 5 ("ahatape" header file):

```
# Use this record on a UNIX system to read directly
# from a 9-track AHA DB distribution tape with
# 4096-byte blocks.  The tape must be positioned
# to the beginning of the ECG  data file before
# using this record.

ahatape 2 250
/dev/nrmt0 16 0 12 0 0 0 4096
/dev/nrmt0 16 0 12 0 0 0 4096
```

As in the previous example, both signals are associated with the same file; in this case, the file is */dev/nrmt0*, the non-rewinding raw 9-track tape drive (on some systems, the name of this device may differ). The block size must be specified in this case, since I/O to or from a raw device (character special file) is not buffered by the operating system and must be performed in the units appropriate to the device (in this case, the tape block size). AHA DB tapes written at 1600 bpi contain 4096 bytes per block (i.e., 1024 two-byte samples from each of the two signals).

Example 6 ("multi" header file):

```
multi/3 2 360 45000
100s 21600
null 1800
100s 21600
```

This header file is a sample of a multi-segment record. The first line contains the record name ("multi"), the number of segments (there are 3), the number of signals (2; this must be the same in each segment), the sampling frequency (360), and the total length of the record in sample intervals (45000; this must be the

sum of the segment lengths).

The second line contains the record name ("100s") of the first segment of the record, and its length in sample intervals (21600). The third and fourth lines contain the record names and lengths of the remaining segments. The remaining lines are comments.

Note that a segment may appear more than once in a multi-segment record, as in this sample, and that storage formats may vary between segments (the second segment is a "null" record, containing format 0 "signals", and the others are written in format 8).

This record may be read by any WFDB application built using WFDB library version 9.1 or later; the application need not be aware that this is a multi-segment record. Earlier versions of the WFDB library do not support multi-segment records (or format 0 signals).

Old format

Versions 2.3 through 4.6 of the WFDB library included support for reading header files written in an obsolete format. This support has been removed from WFDB library version 5.0. Obsolete-format header files can be brought up-to-date using *revise* (in the *convert* directory of the WFDB software distribution).

SEE ALSO

annot(5), **signal**(5), **wfdbcal**(5)
WFDB Programmer's Guide

AUTHOR

George B. Moody (george@mit.edu)

NAME

signal – WFDB signal file formats

DESCRIPTION

WFDB signal files exist in several formats. Any of these formats can be used for multiplexed signal files, in which samples from two or more signals are stored alternately. See **header**(5) for information on how to identify which of the formats below is used for a particular signal file.

Format 8

Each sample is represented as an 8-bit first difference; i.e., to get the value of sample *n*, sum the first *n* bytes of the sample data file together with the initial value from the header file. When format 8 files are created, first differences which cannot be represented in 8 bits are represented instead by the largest difference of the appropriate sign (-128 or +127), and subsequent differences are adjusted such that the correct amplitude is obtained as quickly as possible. Thus there may be loss of information if signals in another of the formats listed below are converted to format 8. Note that the first differences stored in multiplexed format 8 files are always determined by subtraction of successive samples from the same signal (otherwise signals with baselines which differ by 128 units or more could not be represented this way).

Format 16

Each sample is represented by a 16-bit two's complement amplitude stored least significant byte first. Any unused high-order bits are sign-extended from the most significant bit. Historically, the format used for MIT–BIH and AHA database distribution 9-track tapes was format 16, with the addition of a logical EOF (octal 0100000) and null-padding after the logical EOF.

Format 24

Each sample is represented by a 24-bit two's complement amplitude stored least significant byte first.

Format 32

Each sample is represented by a 32-bit two's complement amplitude stored least significant byte first.

Format 61

Each sample is represented by a 16-bit two's complement amplitude stored most significant byte first.

Format 80

Each sample is represented by an 8-bit amplitude in offset binary form (i.e., 128 must be subtracted from each unsigned byte to obtain a signed 8-bit amplitude).

Format 160

Each sample is represented by a 16-bit amplitude in offset binary form (i.e., 32,768 must be subtracted from each unsigned byte pair to obtain a signed 16-bit amplitude). As for format 16, the least significant byte of each pair is first.

Format 212

Each sample is represented by a 12-bit two's complement amplitude. The first sample is obtained from the 12 least significant bits of the first byte pair (stored least significant byte first). The second sample is formed from the 4 remaining bits of the first byte pair (which are the 4 high bits of the 12-bit sample) and the next byte (which contains the remaining 8 bits of the second sample). The process is repeated for each successive pair of samples. Most of the signal files in PhysioBank are written in format 212.

Format 310

Each sample is represented by a 10-bit two's-complement amplitude. The first sample is obtained from the 11 least significant bits of the first byte pair (stored least significant byte first), with the low bit discarded. The second sample comes from the 11 least significant bits of the second byte pair, in the same way as the first. The third sample is formed from the 5 most significant bits of each of the first two byte pairs (those from the first byte pair are the least significant bits of the third sample). Note that the unused bit in each byte pair is set to zero when using the WFDB library to write a format 310 signal file. The entire process is then repeated for each successive set of three samples.

Format 311

Each sample is represented by a 10-bit two's-complement amplitude. Three samples are bit-packed into a 32-bit integer as for format 310, but the layout is different. Each set of four bytes is stored in little-endian

order (least significant byte first, most significant byte last). The first sample is obtained from the 10 least significant bits of the 32-bit integer, the second is obtained from the next 10 bits, the third from the next 10 bits, and the two most significant bits are unused (note that these bits are set to zero when using the WFDB library to write a format 311 signal file). This process is repeated for each successive set of three samples.

If the format specifies a number of bits per sample that exceeds the number of bits in a **WFDB_Sample**, the excess high bits are not read on input, and they are replaced by zeroes on output. Currently, this can happen only when using formats 24 or 32 on a 16-bit platform (unusual except for embedded processors); in this case, the **WFDB_Sample** data type may be redefined as **long** (in *wfdb/wfdb.h*) before compiling the WFDB library and applications, to enable full-precision processing of signals in all formats. This is not done by default since it would increase memory and computational requirements unnecessarily in embedded applications that do not require 24- or 32-bit precision.

SEE ALSO

annot(5), **header**(5), **wfdbcal**(5)
WFDB Programmer's Guide

AUTHOR

George B. Moody (george@mit.edu)

NAME

wfdbcal – WFDB calibration file format

DESCRIPTION

Programs compiled using the WFDB library (see **wfdb**(3)) require calibration data in order to convert between sample values (expressed in analog-to-digital converter units, or adus) and physical units. Calibration files specify the physical characteristics of calibration pulses that may be present in various types of signals, and specify customary scales for plotting these signals. **calsig**(1) reads the signal file(s) for a record, measures the size of the calibration pulses it finds in adus, and uses specifications from a calibration file to determine adu-to-physical unit conversion parameters, the 'gain' and 'baseline' fields that it writes back into the header file for the record. Other programs, such as **pschart**(1), make use of the 'gain' and 'baseline' fields from the header file to determine how to convert adus into physical units, and use customary scale specifications from a calibration file to determine how to convert physical units into units of length on a printed page or on-screen. Most users will find that a single calibration file, perhaps a system-wide default, can be used with all of their WFDB records.

Calibration files are line-oriented text files. Lines are separated by a carriage-return/line-feed pair. Each type of signal to be calibrated is described by a one-line entry. The format of each entry is:

DESC<*tab*>**LOW HIGH TYPE SCALE UNITS**

where **DESC** is a string, possibly containing embedded spaces but not tabs, taken from the signal description field of the header file entry for signals of the desired type; **LOW** and **HIGH** are the physical measurements that correspond to the low- and high-amplitude phases of the calibration pulse; **TYPE** specifies the shape of the calibration pulse ('sine', 'square', or 'undefined'); **SCALE** specifies the customary scale in physical units per centimeter; and **UNITS** is a string (without embedded whitespace) that specifies the physical units of the signal (e.g., 'mV', 'mmHg', 'degrees_Celsius'). If **LOW** is '-', the signal is AC-coupled, and **HIGH** is taken as the peak-to-peak amplitude of the calibration pulse. **LOW** *must* be defined (i.e., must not be '-') for DC-coupled signals. If **HIGH** is '-', the size of the calibration pulse is undefined.

Lines that begin with '#', empty lines, and improperly formatted lines are treated as comments and ignored.

The WFDB library function *getcal*, used by programs such as **calsig**(1), **psfd**(1), and **wave**(1) to obtain calibration data from a calibration file, returns the first entry that matches a signal's description and units. A calibration file entry is considered to match a signal if the **DESC** field is either an exact match or a prefix of the signal description as given in the header file, and if the **UNITS** field in the calibration file is an exact match of the units field in the header file. By making use of these two rules, it is possible to write a calibration file that contains entries for several specific cases followed by a 'catch-all' case for which the **DESC** field contains only the common prefix.

Note that **SCALE** specifications are advisory, not mandatory. The intended use of **SCALE** is to specify the customary size for signals, and the relative sizes of signals of varying types. When determining a **SCALE** for a signal type for which there is no customary scale, a good rule of thumb is that the typical short-term range of variation of the plotted signal should be on the order of one centimeter; keep in mind that it may be useful to make measurements on plots, however, and choose a scale that makes such measurements easy to perform. Programs that draw signals at non-standard scales should generally adjust the scales for all signals by the same factor, unless the user specifies otherwise.

Examples

```
# A simple example of a WFDB calibration file
ECG     - 1 sine 1 mV
NBP     0 100 square 100 mmHg
IBP     0 - square 100 mmHg
Resp    - - undefined 1 l
```

In this example, the first line is a comment. The second line specifies that signals whose descriptions begin with 'ECG' are AC-coupled, have units of millivolts (mV), have 1 mV (peak-to-peak) sine-wave calibration signals, and are customarily drawn at a scale of 1 mV/cm. The third line specifies that signals of the 'NBP' type are DC-coupled, have units of millimeters of mercury (mmHg), square-wave calibration signals that go from 0 to 100 mmHg, and are customarily drawn at a scale of 100 mmHg/cm. The fourth line specifies that signals of the 'IBP' type are DC-coupled (since **LOW** is specified), also have units of mmHg, and are

customarily drawn at a scale of 100 mmHg/cm, but that calibration pulses may vary in amplitude. The last line specifies that 'Resp' signals are AC-coupled (since **LOW** is not specified), have calibration pulses of variable size and shape, and have units of liters [l].

An entry of the form:

ECG lead I - 1 sine 1 mV

matches 'ECG lead II' as well as 'ECG lead I', because of the prefix rule (see above). If 'ECG lead I' and 'ECG lead II' were to require different calibrations for some reason, an entry of the form:

ECG lead II - 2 sine 1 mV

should be inserted *before* the entry for 'ECG lead I'.

Programs that display time series extracted from annotation files (e.g., **wave**(1), which can display the sequence of 'num' fields in an annotation file as a signal) can use calibration records to choose an ordinate scale. These records can be included in the calibration file, with annotator names used in place of the signal type, and 'units' as the units type. An entry with signal type "ann" can be used as a default for calibrating data from files whose annotator names do not have entries. For example, the default calibration file contains these entries:

edr - - undefined 200 units

ann - - undefined 100 units The first specifies that data from 'edr' annotation files are to be displayed at a nominal 200 units per centimeter. The second specifies that files from other types of annotation files are to be displayed at 100 units per centimeter.

ENVIRONMENT

Programs compiled with the WFDB library use the environment variable **WFDBCAL** to determine the name of the calibration file. Calibration files must be located in one of the directories named by the WFDB path (see *setwfdb*(1)).

SEE ALSO

calsig(1), **setwfdb**(1), **annot**(5), **header**(5). **signal**(5)
WFDB Programmer's Guide

AUTHOR

George B. Moody (george@mit.edu)

Installing the WFDB Software Package

George B. Moody
Harvard-MIT Division of Health Sciences and Technology, Cambridge, MA, USA

This appendix briefly describes how to install the WFDB Software Package on a new system. The package includes C-language sources for the WFDB library and for most of the applications described in this manual, sources for this manual, the *WFDB Programmer's Guide*, and the *WAVE User's Guide*, and a one-minute sample record (`100s`).

The latest version of the package can always be downloaded in source form from `http://physionet.org/physiotools/wfdb.shtml`, the WFDB home page on PhysioNet. Binaries for popular operating systems and development snapshots are also usually available there.

The process for installing the package is the same on all platforms, and is documented in detail in the quick-start guides for the popular platforms that can be found on the WFDB home page. In brief:

1. *Install any prerequisites needed for your platform.* These include `gcc` (the GNU Compiler Collection), related software development tools such as `make`, a supported HTTP client library (either `libcurl` or `libwww`; this can be omitted if NETFILES support is not desired), the XView libraries (needed for WAVE only), and X11 (needed by XView). All of these components are free (open-source) software available for all popular platforms, including GNU/Linux, Mac OS X, MS Windows, and Unix. The quick start guides list recommended packages and where to find them.

2. *Download and unpack the WFDB Software Package.* Versions for all platforms are built from a single package of portable sources; the most recent package is always available at `http://physionet.org/physiotools/wfdb.tar.gz`.

3. *Configure the package for your system.* The `configure` script creates a customized building procedure for your system and allows you a few choices about where to install the package.

4. *Make and verify a test build.* The package includes a set of test scripts that are run to verify basic operations of the WFDB library and many of the applications, permitting them to be tested before installation.

5. *Make, install, and test a final build.*

See the quick start guide for your platform for detailed step-by-step instructions.

Important: Although you may be able to compile the WFDB Software Package using a proprietary compiler, this is *not supported*.

Evaluating ECG Analyzers

George B. Moody
Harvard-MIT Division of Health Sciences and Technology, Cambridge, MA, USA

Summary

This paper describes how to evaluate an automated ECG analyzer using available annotated ECG databases and software, in compliance with standard evaluation protocols. These protocols have been adopted as parts of the *American National Standard for Ambulatory Electrocardiographs* (ANSI/AAMI EC38:1998, and its predecessor, ANSI/AAMI EC38:1994), and the *American National Standard for Testing and Reporting Performance Results of Cardiac Rhythm and ST Segment Measurement Algorithms* (ANSI/AAMI EC57:1998). They include earlier evaluation protocols developed for an AAMI Recommended Practice, *Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms* (AAMI ECAR, 1987). It will be most useful to readers who plan to use the suite of evaluation software included in the WFDB Software Package (`http://www.physionet.org/physiotools/wfdb.shtml`); this suite of software includes the reference implementations of the evaluation protocols specified in EC38 and EC57.

1 Introduction

Continuous monitoring of the electrocardiogram in both inpatients and ambulatory subjects has become a very common procedure during the past thirty years, with diverse applications ranging from screening for cardiac arrhythmias or transient ischemia, to evaluation of the efficacy of antiarrhythmic drug therapy, to surgical and critical care monitoring. Since the first intensive care units were established in the 1960s, the need for automated data reduction and analysis of the ECG has been apparent, motivated by the very large amount of data that must be analyzed (on the order of 10^5 cardiac cycles per patient per day). As clinical experience has led to the identification of more and more prognostic indicators in the ECG, clinicians have demanded and received increasingly sophisticated automated ECG analyzers. The early heart rate monitors rapidly evolved into devices that were designed first to detect ventricular fibrillation, then other "premonitory" ventricular arrhythmias. Many newer devices attempt to detect supraventricular arrhythmias and transient ischemic ST changes.

Visual analysis of the ECG is far from simple. Accurate diagnosis of ECG abnormalities requires attention to subtle features of the signals, features that may appear only rarely, and which are often obscured by or mimicked by noise. Diagnostic criteria are complicated by inter- and intra-patient variability of both normal and abnormal ECG features. Given these considerations, it is not surprising that developers are faced with a difficult task in the design of algorithms for automated ECG analysis, and that the results of their efforts are imperfect. Certain parts of the problem — QRS detection in the absence of noise, for example — are well-solved by most current algorithms; others — detection of supraventricular arrhythmias, for example — remain exceedingly difficult. Just as we may find it easiest to analyze "textbook" examples, automated ECG analyzers may perform better while analyzing the recordings used during their development than when applied to "real-world" signals.

Since automated ECG analyzers vary in performance, and since their performance is dependent on the characteristics of their input, quantitative evaluations of these devices are essential in order to assess the usefulness of their outputs. At one extreme, a device's outputs in the context of a particular type of signal may be so unreliable as to be worthless; unfortunately, the other extreme — an output so reliable it can be accepted uncritically — is not a characteristic of any existing monitor, nor can it be expected in the future.

1.1 ECG Databases

Several databases of ECG recordings are generally available for evaluating ECG analyzers. They serve several important needs:

- They contain *representative* signals. Wide variations in ECG characteristics among subjects severely limit the value of synthesized waveforms for testing purposes. Realistic tests of ECG analyzers require large sets of "real-world" signals.
- They contain *rarely observed but clinically significant* signals. Although it is not particularly difficult to obtain recordings of common ECG abnormalities, often those that are most significant are rarely recorded. Both developers and evaluators of ECG analyzers need examples of such recordings.
- They contain *standard* signals. System comparisons are meaningless unless performance is measured using the same test data in each case, since performance is so strongly data-dependent.
- They contain *annotated* signals. Typically, each QRS complex has been manually annotated by two or more cardiologists working independently. The *reference* annotations produced as a result serve as a "gold standard" against which a device's analysis can be compared quantitatively.
- They contain *digitized, computer-readable* signals. It is therefore possible to perform a fully automated, strictly reproducible test in the digital domain if desired, allowing one to establish with certainty the effects of algorithm modifications on performance.

Standards EC38 and EC57 require the use of the following ECG databases:[1]

- **AHA DB**: The American Heart Association Database for Evaluation of Ventricular Arrhythmia Detectors (80 records, 35 minutes each)
- **MIT DB**: The Massachusetts Institute of Technology–Beth Israel Hospital Arrhythmia Database (48 records, 30 minutes each)
- **ESC DB**: The European Society of Cardiology ST-T Database (90 records, two hours each)
- **NST DB**: The Noise Stress Test Database (12 records, 30 minutes each)
- **CU DB**: The Creighton University Sustained Ventricular Arrhythmia Database (35 records, 8 minutes each)

Each of these databases represents a very substantial effort by many workers; in particular, the AHA, MIT, and ESC databases each required more than five years of sustained effort by large teams of researchers and clinicians from many institutions. Nevertheless, it should be recognized that even these databases do not fully represent the variety of "real-world" ECGs observed in clinical practice. Although these databases permit standardized, quantitative, automated, and fully reproducible evaluations of analyzer performance, it is risky to extrapolate from the results of such evaluations to expectations of real-world performance. Such extrapolations can be particularly error-prone if the evaluation data were also used for development of the analysis algorithm, since the algorithm may have been (perhaps unintentionally) "tuned" to its training set. It should also be noted that the first four of the databases listed above were obtained from Holter ECG recordings; although the frequency response of the Holter recording technique is not usually a limiting factor in the performance of an ECG analyzer, it may tend to favor devices that are designed to analyze Holter recordings over devices that have been designed to analyze higher-fidelity input signals.

1.2 Evaluation Protocols

Between 1984 and 1987, the Association for the Advancement of Medical Instrumentation (AAMI) sponsored the development of a protocol for the use of the first two of these databases, which was published as an AAMI Recommended Practice.[2] Between 1990 and 1998, the ambulatory ECG subcommittee of the AAMI ECG committee

[1] Sources: ECRI, 5200 Butler Pike, Plymouth Meeting, PA 19462 USA (AHA DB); PhysioNet (http://physionet.org/) (MIT, NST, CU DB; and ESC DB for non-commercial use); Alessandro Taddei, CNR Institute of Clinical Physiology, G. Pasquinucci Heart Hospital, via Aurelia Sud, 54100 Massa, Italy (ESC DB for commercial use).

[2] *Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms*. Publication AAMI ECAR (1987); succeeded by ANSI/AAMI EC57:1998, available from AAMI, 1110 N Glebe Road, Suite 220, Arlington, VA 22201 USA.

developed and revised a standard for ambulatory ECG monitors, significant portions of which address the issue of the accuracy of automated analysis performed by some of these devices.[3] The ambulatory ECG standard EC38:1998, and the "testing and reporting performance results" standard EC57:1998, build on the evaluation protocol adopted for the earlier Recommended Practice (ECAR), incorporating provisions for the use of all five of the databases listed above, with extensions for assessing detection of supraventricular arrhythmias and transient ischemic ST changes. The standard breaks new ground in establishing specific reporting requirements for the performance of automated ECG analyzers on standard tests using the databases listed above.

A significant constraint imposed on evaluators by the EC38 standard is that they must obtain annotation files containing the analysis results of the device under test. Although the device itself need not produce these files, EC38 specifically requires that they be produced by an automated procedure, which must be fully disclosed. The intent of this requirement is to permit reproducible independent evaluations in which neither the proprietary data of the developers (the analysis algorithms) nor that of the evaluators (the test signals and reference annotations) need necessarily to be disclosed. By defining the interface between the developer and the evaluator to be the annotation file, the responsibilities of each party are clearly defined: the developer must make certain that the device's outputs are recorded in the annotation file in the manner intended by the developer, but in the language of the standard; the evaluator must make certain that the algorithms used to compare the device's annotation files with the reference annotation files conform to the specification of the standard. The format and content of these annotation files is specified in detail below. For many existing devices, it may be difficult or impossible to obtain such annotation files without the cooperation of the developers. Newly-designed devices should incorporate the necessary "hooks" for producing annotation files.

1.3 Software to Support Evaluations

This paper describes a suite of programs that support evaluations of automated ECG analyzers in accordance with the methods described in the EC38 and EC57 standards (as well as those in the earlier ECAR Recommended Practice). These methods are sufficiently complex that the development of such a suite of programs is not an afternoon's work. By making generally available reference implementations of the evaluation algorithms, much needless duplication of effort may be avoided. By circulating them in source form to other users, we may hope to find and correct any bugs, with the eventual result that evaluators of devices should not have to bear the burden of evaluating the evaluation technique itself. By using them for evaluations, any ambiguities in the English specification of the evaluation algorithms are resolved in a consistent manner for each device tested. These programs are written in C and run under MS-DOS or UNIX. They have been made available as part of the WFDB Software Package. In this paper, the names of these programs are printed `like this`.

2 Evaluating an ECG Analyzer

The major task facing an evaluator is that of presenting the reference signals to the device under test, and collecting annotation files from the device. The details of this task will vary for each device, but a few general hints are given below. A second task, that of obtaining reference heart rate measurements, should be a much simpler job. Once all of this information has been gathered, the remaining work required — that of comparing the device's analysis against the "gold standard" — can be performed automatically.

2.1 Presenting Signals to the Analyzer

Two distinctly different types of tests are possible. If the device can accept digital inputs, the reference signals can be supplied in that form (perhaps after resampling with `xform` to convert the digitized samples to the expected sampling frequency and numerical range, and possibly with additional digital signal processing to simulate the signal conditioning normally performed by the device's front-end data acquisition hardware). The primary advantage of testing in the digital domain is that the test is (or should be) strictly reproducible, since no noise or additional quantization error can be introduced in this way. This method usually avoids the issue of synchronization of the test annotations with the reference signals discussed below.

[3] *American National Standard for Ambulatory Electrocardiographs*. Publication ANSI/AAMI EC38:1998; available from AAMI (address above).

Testing in the analog domain requires that analog signals be recreated from the digital signals. (It should be noted that even the analog versions of the MIT and AHA databases that have been available in the past were recreated from the digitized signals by the database developers.) The advantage of this approach is that it exercises the entire system, including the front-end data acquisition hardware. It is often difficult, however, to establish synchronization between the signal source and the analyzer, needed in order to permit comparisons of annotations. One way of dealing with this problem is to arrange for the analyzer's sampling clock to trigger the digital-to-analog converter used to recreate the analog signals, or to arrange for an external clock to trigger both D/A conversion in the playback system and A/D conversion in the analyzer. Another method is to begin and end the signal generation process by delivering signals from the analyzer to the playback device, and recording the analyzer's clock time at the times of the signals; assuming that both the analyzer and the playback device have stable clocks, event times in the analyzer's frame of reference can be converted to database sample numbers by linear interpolation. The WFDB software package includes a program (`sample`) that uses a Microstar DAP 2400-series analog interface board[4] and an MS-DOS PC to recreate analog signals from digital database records on CD-ROMs or magnetic disk files.

2.2 Obtaining Test Annotation Files

For any ambulatory ECG monitor that incorporates automated analysis functions, the EC-38 standard requires the manufacturer to implement and disclose a method for producing test annotation files. Independent evaluators should seek assistance from the manufacturer in any case, since the manufacturer's interpretation of the device's outputs in the language of EC-38 is definitive (in effect, the annotation file generation technique becomes part of the system under test). Note that generation of annotation files need not be synchronous with data acquisition; a device might conceivably store all of the necessary data until the end of the test, and only then write the file. Neither does the standard require that an annotation be determined within any fixed amount of time, as would be expected of devices designed to trigger pacing, for example. Furthermore, EC-38 specifically allows for the possibility that the device under test might not produce the annotation file directly. If any external hardware or software is required to do so, however, it must be made generally available or specified in sufficient detail by the manufacturer to permit an independent evaluator to obtain test annotation files.

Annotation files contain a label (an annotation) for each beat and for certain other features of the signals, such as rhythm and ST changes. Annotations are stored in time order in annotation files. The "time" of an annotation is that of the sample in the signal file with which the annotation is associated.[5] The WFDB library (included in the WFDB software package) includes C-callable functions (`getann` and `putann`) for reading and writing annotations. In a C program, annotations appear as data structures containing a 32-bit `time` field together with a pair of 8-bit fields that encode the annotation type and sub-type (`anntyp` and `subtyp` [sic], respectively), and a variable-length `aux` field usually used to store text. In annotation files, these annotation structures are usually stored in a variable-length bit-packed format averaging slightly more than 16 bits per annotation.[6]

Test annotation files may include the following:

- *Beat annotations*. These need not coincide precisely with the reference beat annotations, since the evaluation protocol allows a time difference of up to 150 ms between each pair of matching beat annotations. All beat annotations are mapped during the evaluation process into the set { N, V, F, S, Q } (corresponding to normal, ventricular ectopic, ventricular fusion, supraventricular ectopic, and unclassifiable or paced beats respectively); devices need not be capable of producing all of these annotations, but any beat annotations that they do produce will be translated into one of these types. The standard specifies the mapping used for the `anntyp` values defined in `<wfdb/ecgcodes.h>`. (This file is included in the WFDB Software Package.) Any beat annotations that appear in the first five minutes of a record (the "learning period") are ignored in the evaluation process. The remainder of the record (the "test period") must be fully annotated. Note in particular that the last beat of some records may be very close to the last sample; since the analyzer may reach the end

[4] Source: Microstar Laboratories, `http://www.mstarlabs.com/`. External analog anti-aliasing filters (to reduce "staircasing") and attenuators (to obtain patient-level signals) may also be required, depending on the system to be evaluated. DAP boards can also be used with `sample` to create new database records.

[5] Times in annotation and signal files are usually expressed as *sample numbers* (the number of samples in the signal file that precede the sample in question).

[6] Test annotations that include heart rate or ST measurements require substantially more storage. `getann` and `putann` can also use the original AHA DB format (containing fixed-length annotations, 16 bytes each), but this format should not be used for evaluations of devices that incorporate ST analysis functions, since the space available for the `aux` data is too small to store ST measurements.

of the input signals before producing an annotation for the last beat, it may be necessary to "pad" the input data for a few seconds at the end of the record to permit the analyzer to emit its final beat annotation.

- *Shutdown annotations*. If the device suspends its analysis because of poor signal quality or for any other reason, it should mark the periods during which analysis is suspended. The evaluation software tallies beats missed during such periods separately from beats missed at other times. The beginning of each period of shutdown may be marked using a `NOISE` annotation with `subtyp` $= -1$, and the end of each period of shutdown may be marked using a `NOISE` annotation with `subtyp` $= 0$ (see the source for `bxb` for notes on other acceptable methods of marking shutdown).

- *Ventricular fibrillation annotations*. The beginning and end of each detected episode of ventricular fibrillation should be marked using `VFON` and `VFOFF` annotations.

- *Other rhythm annotations*. These should include `RHYTHM` annotations marking the beginning and end of each detected episode of atrial fibrillation. The beginning of each episode should be marked with an "`(AFIB`" rhythm annotation, i.e., an annotation with `anntyp` = `RHYTHM` and `aux` = `"\05(AFIB"`, where "`\05`" is C notation for a byte with the value 5 (ASCII control-E). Non-empty `aux` fields always begin with a byte that specifies the number of data bytes that follow; in this case, the five characters (`( A F I B`) of the string. The end of each episode should be marked with any other rhythm annotation (for example, `"\02(N"`).

- *Heart rate measurements*. Each type of heart rate measurement (including any heart rate or RR interval variability measurements) made by the device under test should be assigned a measurement number, m, between 0 and 127. A `MEASURE` annotation should be recorded for each heart rate measurement, with `subtyp` $= m$ and with the measurement in the `aux` field, as an ASCII-coded decimal number.

- *ST deviation measurements*. If available, these should be provided in the `aux` fields of beat annotations, as ASCII-coded decimal numbers indicating the deviations in microvolts from reference levels established for each signal from the first 30 seconds of each record. For example, "`25 -104`" indicates a 25 μV elevation in signal 0 and a 104 μV depression in signal 1. If ST measurements are omitted from any beat annotation, the evaluation software assumes they are unchanged from their previous values.

- *Ischemic ST change annotations*. These `STCH` annotations should mark the beginning and end of each detected episode of ischemic ST change. ST change annotations have additional information in the `aux` field as for rhythm annotations: the beginning of each episode is marked by an "`(ST`*ns*" annotation, and the end of each episode by a "`ST`*ns*`)`" annotation, where *n* indicates the signal affected ("`0`" or "`1`"), and *s* indicates ST elevation ("+") or depression ("−"). *n* may be omitted if the episode detection criteria depend on features of both signals. The extremum of each episode may optionally be marked with an "`AST`*nsm*" annotation, where *n* and *s* are defined as above, and *m* is the ST deviation in microvolts, relative to a reference level established as above.

- *Comment annotations*. Annotations with `anntyp = NOTE` and any desired string data in `aux` may be included anywhere in an annotation file. `NOTE` annotations are ignored by the standard evaluation software; they may be used, for example, to record the values of internal algorithm variables for debugging purposes.

Note that only beat annotations are absolutely required in test annotation files. ST deviation measurements within beat annotations, and the other types of annotations listed above, only need to be recorded for devices that are claimed by their manufacturers to provide optional features for detection of ventricular or atrial fibrillation, measurement of ST deviations, or detection of ischemic ST changes.

If the time units in the test annotation files are not the same as those in the reference annotation files (for example, because `xform` was used to change the sampling frequency of the signal files in a digital-domain test), the time units must be rescaled before proceeding with the comparison. This may be done by using `xform` to rewrite the test annotation files with the original sampling frequency.[7]

[7]The obvious alternative, using `xform` to rewrite the reference annotation files at the time the signal files are resampled, should not be used in a formal evaluation. Because of the possibility that resampling the reference annotation files might result in moving reference annotations into or out of the test period, or changing the lengths of episodes, doing so might produce results that could not be directly compared with those obtained in a standard evaluation.

Details of the ST deviation measurement and episode detection criteria used in producing the reference annotation files for the ESC database may be found in several sources.[8] Note, however, that many techniques for measuring ST deviation and for detecting transient ischemic ST changes are possible, and that to date the best evaluation results have been obtained for analyzers using criteria that do not attempt to mimic those used by the human experts who annotated the database.

2.3 Obtaining Reference Heart Rate Data

The final step of preparation for the evaluation is to process the reference annotation files to obtain reference heart rate annotation files. These files must contain heart rate measurement annotations with the same measurement numbers assigned as for the test heart rate annotations; they need not necessarily contain beat or other annotations from the reference annotation files. Quoting from EC38,

> To evaluate the accuracy of heart rate measurement, the evaluator shall implement and disclose a method for obtaining heart rate measurements using the reference annotation files (the 'reference heart rate'). This method need not be identical to the method used by the device under test, but in general it will be advantageous if it matches that method as closely as possible.

It will generally be in the manufacturer's interest to provide a program for generating reference heart rate annotation files, to avoid the need for an independent evaluator to do so, with a likely result of less than optimal agreement with the test heart rate measurements. The WFDB software package includes a sample implementation of such a program (`examples/refhr.c`); note that it will need to be customized for each device to be tested.

Note that measurement errors are normalized by the mean value of the reference measurements in each record. Be certain that this mean value cannot be zero![9]

3 Comparing Annotation Files

Once the test annotation files and the reference heart rate annotation files have been obtained, the remainder of the evaluation procedure is straightforward. All of the information needed to characterize the analysis performed by the device under test is encoded in the test annotation files; similarly, all of the information needed to characterize the actual contents of the test signals is encoded in the reference annotation and reference heart rate annotation files. The evaluation procedure thus entails comparison of the test and reference annotation files for each record.

Four programs are provided in the WFDB Software Package for this purpose:

- `bxb` compares annotation files beat by beat; its output includes QRS, VEB, and (optionally) SVEB sensitivity and positive predictivity, as well as RR interval error and shutdown statistics.
- `rxr` compares annotation files run by run; its output includes ventricular (and, optionally, supraventricular) ectopic couplet, short run (3–5 beats), and long run (6 or more beats) sensitivity and positive predictivity.
- `epicmp` compares annotation files episode by episode; its output includes ventricular fibrillation, atrial fibrillation, and ischemic ST detection statistics as well as comparisons of ST deviation measurements.
- `mxm` compares measurements from a test annotation file and a reference heart rate annotation file; its output indicates measurement error.[10]

The WFDB Software Package also includes three related programs:

[8] See, for example, the *European ST-T Database Directory*, pp. vi-vii, supplied with the ESC DB; or Taddei, A., et al., "The European ST-T database: development, distribution, and use", *Computers in Cardiology* **17**:177-180 (1990).

[9] For certain types of HRV or RRV measurements (though not for heart rate measurements), this is a potential problem. One solution is to add a small positive offset to any measurement with an expected zero mean. It is within the letter, though not the spirit, of the standard protocol, to add a very large number in such a case, so as to make the error percentage arbitrarily small. The mean value of the reference measurements must be reported; this should serve as a disincentive to this sort of creative abuse of the standard. An honest approach might be to add an offset on the order of the expected standard deviation of the individual measurements.

[10] `mxm` is not restricted to comparison of heart rate measurements; if other types of measurements are available, they may be compared in the same manner as heart rates by `mxm`.

- `sumstats` reads certain output files generated by `bxb`, `rxr`, `epicmp`, and `mxm`, and calculates aggregate statistics for a set of records.
- `plotstm` generates scatter plots of ST deviation measurements collected by `epicmp`.
- `ecgeval` automates the entire comparison procedure by running `bxb`, `rxr`, `epicmp`, and `mxm` for each record, collecting their output, then running `sumstats` (and optionally `plotstm`), and finally printing the results.

To obtain a concise summary of how to use any of these programs, including a list of any command-line options, simply run the program without any command-line arguments. Refer to the *WFDB Applications Guide*, which accompanies the WFDB Software Package, for details.

In most cases, it will be easiest to collect all of the annotation files before beginning the comparison, and then to perform the comparison by typing:

```
ecgeval
```

The program asks for the test annotator name, the names of the databases used for testing, and what optional detector outputs should be evaluated.

Only the statistics required by EC38 and EC57 are reported by `ecgeval`. If more detailed evaluation data are needed, it will be necessary to run `bxb`, `rxr`, etc., separately. If file space is extremely limited, it may be necessary to delete each test annotation file after it has been compared against the reference file, before the next test annotation file can be created; in such cases, it may also be necessary to prompt the user to change media containing signal or reference annotation files, or to reset the device under test before beginning each record. Optionally, `ecgeval` can generate a script (batch) file of commands, which can be edited to accommodate special requirements such as these.

For example, suppose we have obtained a set of test annotation files with the annotator name "`yow`", which we wish to compare against the reference annotation files (annotator name "`atr`")[11] and reference heart rate annotation files (annotator name "`htr`"). The portion of the evaluation script generated by `ecgeval` for MIT DB record 100 is:

```
bxb -r 100 -a atr yow -L bxb.out sd.out
rxr -r 100 -a atr yow -L vruns.out sruns.out
mxm -r 100 -a htr yow -L hr0.out -m 0
epicmp -r 100 -a atr yow -L -A af.out
    -V vf.out -S st.out stm.out
```

(The last two lines shown above form a single command. The `mxm` command gathers statistics on measurement number 0; if other heart rate measurements are defined, `mxm` should be run once for each such measurement, substituting the appropriate measurement numbers for `0` in the output file name, `hr0.out`, and the final argument.) Statistics for the remainder of the MIT DB are obtained by repeating these commands, substituting in each the appropriate record names for `100`. Once these commands have been run for all of the records, the record-by-record statistics will be found in nine files (`bxb.out`, `sd.out`, `vruns.out`, `sruns.out`, `hr0.out`, `af.out`, `vf.out`, `st.out`, and `stm.out`). The first eight of these files contain one line for each record.[12] `sumstats` can read any of these files, and calculates aggregate performance statistics; to use it, type "`sumstats` *file*", where *file* is the name of one of these files. The output of `sumstats` contains a copy of its input, with aggregate statistics appended to the end. Typically this output might be saved in a file to be printed later, e.g.,

```
sumstats bxb.out >>report.out
```

A scatter plot of the ST measurement comparisons performed by `epicmp` can be produced using `plotstm`, the output of which can be printed directly on any PostScript printer. For example, to make a plot file for `stm.out`, type:

```
plotstm stm.out >stm.ps
```

[11] Annotation files for any given record are distinguished by annotator names, which correspond to the "extension" of the file name. The reference annotation files supplied with the databases have the annotator name "`atr`" (originally "`atruth`" because "`a`" was intended to indicate the file type, and "`truth`" because ... well, because the annotations are supposed to be The Truth).

[12] `stm.out` contains one line for each ST deviation measurement that was compared; in this example, `stm.out` would be empty since the reference annotation files of the MIT DB do not contain ST deviation measurements.

4 Studying Discrepancies

Having conducted an evaluation as described above, a common question is "what were the errors?" `bxb` and `rxr` can help answer such questions.

`bxb` can generate an output annotation file (with annotator name "`bxb`") in which all matching beat annotations are copied from the test annotation file, and each mismatch is indicated by a `NOTE` annotation, with the `aux` field indicating the element of the confusion matrix in which the mismatch is tallied (e.g., "`Vn`" represents a beat called a VEB by the reference annotator and a normal beat by the test annotator). Programs such as `wave`[13] can be used to search for and display the waveforms associated with the mismatches. To generate an output annotation file, add the `-o` option to the `bxb` command line, as in:

```
bxb -r 100 -a atr yow -L bxb.out sd.out -o
```

A particularly useful way to document an evaluation is to print a full disclosure report with `bxb` output annotations, using the program `psfd` (also included in the WFDB Software Package). This may be accomplished by preparing a file containing a list of the names of the records to be printed (call it `list`), and then using the command:

```
psfd -a bxb list >output.ps
```

The file `output.ps` can be printed on any PostScript printer. Run `psfd` without any arguments for a summary of its (numerous) options; try a short test before making a large set of printouts, which can take a long time.

Both `bxb` and `rxr` accept a `-v` option to run in "verbose" mode, in which each discrepancy is reported in the standard error output. When running `rxr`, this feature is useful for finding missed and falsely detected ectopic couplets and runs.

5 Acknowledgements

Having been involved in the production of most of the databases as well as the design of the evaluation protocols, it has been my privilege to receive the benefits of the sustained contributions of many colleagues who have supported these projects with their dedicated efforts. I would like especially to thank Paul Albrecht, Jim Bailey, Ted Baker, Rich Bowser, Don Brodnick, Jerry Cox, Phil Devlin, Charlie Feldman, Scott Greenwald, Russ Hermes, David Israel, Franc Jager, Carlo Marchesi, Roger Mark, Joe Mietus, Warren Muldrow, Diane Perry, Scott Peterson, Ken Ripley, Paul Schluter, Alessandro Taddei, Roy Wallen, and Cees Zeelenberg.

A Using the AHA Database

Since the AHA DB is not available in the standard PhysioBank format used by all of the other databases, the WFDB Software Package includes a pair of programs that convert files read from AHA DB distribution tapes or floppy disks into files in PhysioBank format. `a2m` converts AHA annotation files, and `ad2m` converts AHA signal files and also generates header (`*.hea`) files. (Run these programs without command-line arguments to obtain instructions on their use.) Using `a2m` and `ad2m`, all 80 AHA DB records can be stored in roughly 130 Mb of disk space (assuming use of the standard 35-minute records). These programs can also reformat old (pre-1989) MIT DB tapes written in the AHA DB distribution format.

It is also possible to read and write AHA tape-format files directly using the WFDB library; refer to the *WFDB Programmer's Guide* for details.

B Noise stress testing

With respect to many tasks performed by an ECG analyzer, dealing with noise is the major problem faced by system designers. Although measurements such as ST deviation may be obtained reliably in clean signals, the presence of noise may render them inaccurate. In some instances, it is sufficient to recognize the presence of noise and either to mark measurements as unreliable or to avoid making measurements altogether. In other cases, excluding noisy data

[13] `wave` (for FreeBSD, Linux, Mac OS X, Solaris, SunOS, and Windows) are included in the WFDB Software Package.

is inappropriate (for example, given the multiple correlations among physical activity, noise, and transient ischemia, excluding noisy signals is likely to introduce sampling bias in an ischemia detector).

It is difficult to measure the effects of noise on an ECG analyzer using ordinary recordings. Even if existing databases include an adequate variety of both ECG signals and noise, the sample size is certainly too small to include all combinations of noise and ECG signals that may be encountered in clinical use. In ordinary recordings, it is difficult or impossible to separate the effects of noise from the intrinsic problems of analyzing clean signals of the same type.

The noise stress test circumvents these problems. By adding noise in calibrated amounts to clean signals, any combination of noise and signal types is possible. Since both the noise-corrupted signal and the clean signal can be analyzed (in separate experiments) by the same analyzer, the effects of noise on the analysis are readily separable from any other problems that may arise while analyzing the clean signals. Finally, since the test can be repeated using different amounts of noise, it is possible to characterize analyzer performance as a function of signal-to-noise ratio.

The major criticisms of the noise stress test are that not all noise is additive, and that the characteristics of the added noise may not perfectly match those of noise observed in clinical practice. These points, though formally irrefutable, do not negate the value of the test. In practice, most of the troublesome noise is additive; thus (given appropriate inputs) the noise stress test can simulate most of the noisy signals of interest. The NST DB includes noise recordings made using standard ambulatory ECG electrodes and recorders, but with electrodes placed on the limbs of active volunteers in configurations in which the subject's ECG is not apparent in the recorded signals. Given the recording technique used, it is not surprising that the characteristics of the recorded noise closely match those of noise in standard ambulatory ECG recordings. Although it may be argued that the particular muscles responsible for the recorded noise might produce different signals than those that generate the EMG present in noisy ECGs, no such differences are apparent from comparisons of either the signals or their power spectra.

The NST DB includes a small set of ECG records with calibrated amounts of added noise. EC38 specifies that performance on these records must be reported, although no specific performance levels are required. Program `nst` can be used to generate additional records for noise stress testing. To do so, choose an ECG record and a noise record (the latter may be `bw`, `em`, or `ma` from the NST DB, or any other available noise recording). Run `nst` and answer its questions to generate a noisy ECG record that may then be used in the same way as any other WFDB record. By default, `nst` adds no noise during the first five minutes of the record, then adds noise for the next two minutes, none for the following two minutes, and repeats this pattern of two minutes of noise followed by two minutes of clean signals for the remainder of the record. The scale factors for the noise, if determined by `nst`, are adjusted such that the signal-to-noise ratios are equal for each signal. The durations of the noisy periods, and the scale factors for each signal, are recorded in a *protocol annotation file*, which is generated by `nst` unless an existing protocol annotation file is supplied as input. To change these parameters, simply edit the protocol annotation file (using, for example, `rdann` to convert it to text form, any text editor to make the modifications, and `wrann` to convert it back to annotation file format), then rerun `nst` using the protocol file to generate a new record.