ECG Database Applications Guide

ninth edition
24 May 1997

George B. Moody

Harvard-MIT Division of Health Sciences and Technology
Biomedical Engineering Center

Copyright © 1992 – 1997 Massachusetts Institute of Technology.

For information on obtaining the most recent versions of the software described in this guide, visit `http://ecg.mit.edu`, or write to:

MIT-BIH Database Distribution
Massachusetts Institute of Technology
77 Massachusetts Avenue, Room 20A-113
Cambridge, MA 02139
USA

An HTML version of this guide is available; point your web browser to `http://ecg.mit.edu/dbag/` to view it. Additional printed copies of this guide are available for US$10 each (including shipping by surface mail) by writing to the address above. Please make checks payable to "Beth Israel Hospital Biomedical Engineering Division."

Permission is granted to make and distribute verbatim copies of this guide provided that the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this guide under the conditions for verbatim copying, provided also that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this guide into another language, under the above conditions for modified versions.

Table of Contents

Section 5: DB file formats

Appendices

INTRODUCTION

Most of this guide consists of UNIX *man* pages that describe the applications included in the MIT DB Software Package. If you are unfamiliar with the format of UNIX *man* pages, it may be helpful to refer to the introduction of a UNIX reference manual for orientation. The descriptions 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.

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.

If you have not used any of these programs before, you should be sure that your environment is set up properly so that DB applications can find their input files. See setdb(1) in this guide for information about doing this; a more detailed discussion may be found in the first chapter of the ECG Database Programmer's Guide, in the section about the database path.

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

record Where this appears, substitute the name of a DB record. 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-DOS 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. Under MS-DOS, furthermore, only the first eight characters of a record name are significant. A record name is *not* a file name. See the *ECG Database Programmer's Guide* for further details on record names.

annotator Where this appears, substitute an annotator name. The annotator name ‘atruth’ (shortened to ‘atr’ under MS-DOS and on CD-ROMs) is used to name the set of *reference annotations* supplied by the database developers. Annotator names may contain letters, digits, and underscores, as for record names. Note that, under MS-DOS, only the first three characters of an annotator name are significant.

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 *ECG Database 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. 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. Where the notation ‘*signal-list*’ (or ‘*signal* ...’) appears in this guide, you may specify more than one signal in any desired order; separate the signal numbers using spaces.

Under UNIX, if the DB Software Package has been installed on your system, you can also access the information contained in this guide using *man* and related programs. 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; point your Web browser to **http://ecg.mit.edu/dbag/** to read it. The HTML version of this guide is also supplied on our CD-ROMs that include the DB Software Package; begin by pointing your browser to **file:///cdrom/dbag/dbag.htm** (under UNIX, if the mount point for your CD-ROM is **/cdrom**), or to **file:///d|/html/dbag.htm** (under MS-DOS, if your CD-ROM is the D: drive).

The notations that appear at the bottom of each page indicate the date when that page was last revised, and (in most cases) the version of the DB 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.

Your comments on this guide, and on the software that it documents, are welcome. Please send them to:

George B. Moody
MIT Room 20A-113
Cambridge, MA 02139
USA

(e-mail: george@hstbme.mit.edu)

NAME

a2m, ad2m, m2a, md2a – utilities for converting between MIT and AHA DB formats

SYNOPSIS

a2m -i *ahafile* **-r** *record* **-a** *annotator* [*options* ...]
ad2m -i *ahafile* **-r** *record* [*options* ...]
m2a -r *record* **-a** *MIT-annotator AHA-annotator* [*options* ...]
md2a -o *ahafile* **-r** *record* [*options* ...]

DESCRIPTION

These programs can be used to convert AHA DB distribution files from half-inch 9-track tape or floppy disks to MIT format (i.e., formats that can be read directly by programs compiled with the *db*(3) library), or to convert files in MIT format into AHA DB tape distribution format (conversion to AHA DB floppy disk distribution format is not supported). All four of these programs print a brief usage summary if invoked with no command-line arguments, or with a **-h** option.

Use *a2m* to convert AHA-format annotation files into MIT 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 or 3, or 2 hours and 30 minutes for type 2).

-t *type* Convert an input file of the specified *type* (0: a file produced by a DB application using *putann* and '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*) 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.

Use *ad2m* to convert AHA-format signal files into MIT format. Options for *ad2m* include:

-c Convert an AHA DB compressed (**.cmp*) floppy disk file (default: convert 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, or the end of the input file, whichever comes first).

Use *m2a* to convert MIT-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.

Use *md2a* to convert MIT-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*.

MIT-format files can be excerpted and reformatted in more generally useful ways using *snip*(1) or *xform*(1).

The shell variable **DB** should be set and exported (see *setdb*(1)).

EXAMPLES

AHA DB floppy disk

To make a version of AHA DB record 1201 in MIT 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 atruth -t 3
```

These commands produce files *1201.dat* (the signal file), *header.1201* (the header file, named *1201.hea* under MS-DOS), and *atruth.1201* (the reference annotation file, named *1201.atr* under MS-DOS), all in the current directory. Run *ad2m* first, so that the new header file is available for the use of *a2m*.

AHA DB short format tape

To obtain the same files given a 'short format' 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 atruth -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 DB 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 format tape

To make a version of the three-hour AHA DB record 1001 in MIT format, given the 'long format' 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 atruth -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 format to short format

To make a version of AHA DB record 1201 in MIT format, given a 'long format' 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 atruth -t 1
```

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

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

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

```
a2m -i 1001.ann -r 1201 -a atruth -t 1
```

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

```
cp header.1001 header.1201
```

or (under MS-DOS) by:

```
copy 1001.hea 1201.hea
```

and edit the copy of the header file, 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 two header and reference annotation files are needed, both versions can share the same signal file, allowing a substantial savings in storage requirements. Note that DB application programs that read the 'short format' 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 DB 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), db(3), header(5)

NAME

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

SYNOPSIS

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

DESCRIPTION

This program implements the beat-by-beat comparison algorithm described in *Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms*, an AAMI Recommended Practice (AAMI ECAR-1987). Using options **-C**, **-L**, or **-S**, *bxb* implements the beat-by-beat comparison algorithm specified by the current American National Standard for ambulatory ECG analyzers (ANSI/AAMI EC38-1994). *bxb* is the reference implementation of these algorithms, and must be used to obtain the beat-by-beat performance statistics cited in EC38 in order to be in compliance with the standard. The EC38 comparison algorithm includes all of the ECAR statistics (so that both sets of requirements are satisfied by using the EC38 algorithm), with additional statistics on detection of supraventricular ectopic beats (SVEBs). 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 EC-38 and in ECAR).

Options include:

-c *file* Append condensed reports (ECAR Table 6 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 (ECAR Tables 7 and 8 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 (ECAR 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 event called 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.

The shell variable **DB** should be set and exported (see *setdb*(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 ECAR or EC-38. 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), epic(1), mxm(1), rxr(1), setdb(1), sumstats(1)
Evaluating ECG Analyzers (in the *ECG Database Applications Guide*)
Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms (publication AAMI ECAR-1987)
American National Standard for Ambulatory Electrocardiographs (publication ANSI/AAMI EC38-1994)
The last two publications are available from AAMI, 3330 Washington Blvd., Suite 400, Arlington, VA 22201 USA.

NAME

calibrate − calibrate signals of a DB record

SYNOPSIS

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

DESCRIPTION

calibrate rewrites the header file for a DB 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, *calibrate* 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, *calibrate* 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, *calibrate* may be run repeatedly with different time intervals and signal lists.

Options include:

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

Calibration files must be located in one of the directories named in the database path (see *setdb*(1)). If the environment variable **DBCAL** 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, *calibrate* 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.

The shell variable **DB** should be set and exported (see *setdb*(1)).

BUGS

Under MS-DOS, this program is known as *calibr8*.

SEE ALSO

setdb(1), dbcal(5)

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)

AVAILABILITY

This program is part of the *psd* package, distributed with the DB Software Package, version 9.3 and later.

NAME

dbcollate − collate DB records into a multi-segment record

SYNOPSIS

dbcollate -i *irec* [*irec ...*] **-o** *orec* [**-a** *annotator*]
dbcollate *orec first last* [**-a** *annotator*]

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.

dbcollate simply constructs an array of segment names, passing it to the DB library function *setmsheader* (see *db*(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

dbcollate xyz 9 12

is equivalent to

dbcollate -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 *dbdesc*(1) to determine the input record length if necessary, then edit the input record header to include this information before using *dbcollate*). 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 both 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.

In most cases, multi-segment records are indistinguishable from single-segment records, from the point of view of applications built using the DB 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 DB library). Note, however, that older applications can generally be updated without source changes simply by recompiling them and linking them with the current DB library.

The shell variable **DB** should be set and exported (see *setdb*(1)).

FILES

header.*orec*	output header file (*orec*.hea under MS-DOS)
header.*irec* or *irec*.hea	input header file(s)

BUGS

Under MS-DOS, this program is known as *dbcoll8*.

SEE ALSO

dbdesc(1), xform(1), db(3), header(5)

AVAILABILITY

This program is included in the DB Software Package, version 9.0 and later.

NAME

dbdesc − read signal specifications

SYNOPSIS

dbdesc *record* [**-readable**]

DESCRIPTION

This program reads specifications for the signals described in the *header* file for *record*. If the **-readable** option is present, **dbdesc** attempts to open the signal files, and it reports only on those that are readable.

The shell variable **DB** should be set and exported (see *setdb*(1)).

FILES

header.*record*	header file
signal files	named in header.*record*

SEE ALSO

setdb(1)

NAME

dbplot − plot DB signals and annotations

SYNOPSIS

dbplot -t*record* [*options*] ...

DESCRIPTION

This program draws DB signals and annotations. The device-independent output of *dbplot* may be piped through *plot*(1) to any of a variety of output devices. Alternatively, if the source is available, *dbplot* may be compiled with any available device-specific *plot*(3) library.

A record name must be supplied in the first argument on the command line. Any legal command may be given on the command line, in a command file, or interactively. Multiple commands may be given on a single line, separated by spaces. The display is redrawn after each command line is processed. Legal commands are:

Search commands:

s*x* The next frame begins with sample *x* (backward searches are allowed unless the signals come from a pipe).

n The next frame begins *n* seconds from the start of the record ...

m:n or *m* minutes and *n* seconds from the start ...

h:m:n or *h* hours, *m* minutes, and *n* seconds from the start.

[**+-**]*n* Any of the commands above may include an optional '+' or '-' to indicate a search relative to the beginning of the current frame.

N (or **V, F, S, ...**)
Annotation mnemonics may be used as search commands. The next frame is centered on the next occurrence of the specified annotation that is not visible in the current frame. Mnemonics are the same as the displayed labels. (See the *ECG Database Programmer's Guide* for a complete list.)

<backspace>
Go backward one frame.

<return>
Go forward one frame (or, if the last search command was an annotation mnemonic, find the next occurrence of the specified annotation)

Zoom commands (VT-52/VT-100 arrow keys):

<up> Zoom into the center of the current frame (*<esc>*A).

<left> Zoom into the left half of the current frame (*<esc>*D).

<right>
Zoom into the right half of the current frame (*<esc>*C).

<down>
Zoom out, redrawing the current frame in the center of the new frame (*<esc>*B).

Options (defaults apply if no qualifier is used):

-a*ann* Display annotations from annotator *ann*. Default: display no annotations.

-c*n* Set compression ratio to *n* (or, if *n* is preceded by an optional **h**, set it so that the output sampling rate is approximately *n* Hz per channel). Default: **h60**.

-d Draw annotation alignment marks as dotted lines. (Invoke again for no marks, and again to restore solid lines.)

-f*file* Redirect the command input to the specified *file*. This command is necessary when displaying a piped record (for which the standard input is used as the sample data file); in this case, **-f/dev/tty** allows normal interactive use. Default: standard input.

-l*n* Draw *n* sets of traces in each frame. Default: 1.

-m*xxx* Specify that the **aux** field of a searched-for annotation must match *xxx*. Default: no match needed.

-p*n* If *n* > 0, pause before erasing (useful if a command file is specified via **-f**). If *n* < 0, pause after erasing (needed for Tektronix 4010 terminals). Default: 0 (no pause).

-r*n* Set plot resolution such that *n*-bit signals will fill the plot area. Default: as given for the current record.

-t*trecord*
Display data from the specified record.

-w*n* Set the frame width to *n* seconds (or, if *n* is preceded by an optional **s**, *n* samples). Default: 5 seconds.

-S Display the **subtyp** field of each annotation.

-C Display the **chan** field of each annotation.

-N Display the **num** field of each annotation.

-A Display the **aux** field of each annotation.

Annotation fields are shown in the order given above. Invoke the **-S, -C, -N,** and **-A** options a second time to cancel.

-F*n* Redefine sampling frequency as *n* Hz. Default: as given for the current record.

-R*h:m:s*
Set the reference time (the clock time in hours, minutes, and seconds corresponding to the beginning of the record).

-T*xxx* Change the title drawn above each frame to *xxx*. Default: restore the standard title.

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

dbtool(1), pschart(1), rdann(1), rdsamp(1), setdb(1), view(1), wave(1)
bin/plot (a shell script; examine it to see which devices are supported)

NAME

dbtool – DB browser for SunView

SYNOPSIS

dbtool -r *record* [**-a** *annotator*]

DESCRIPTION

If you have access to an X Window System display (X11R3 or later), use *wave*(1) rather than *dbtool*. *Wave* is a substantially rewritten and greatly enhanced version of *dbtool*.

dbtool creates a SunView window for viewing the specified *record*. The mouse is used to control the display interactively. By clicking (with the left mouse button) on the ‘Record’, ‘Annotator’, ‘Start time’, ‘End time’, or ‘Find’ fields it is possible to enter new values from the keyboard. Press RETURN when you wish *dbtool* to redraw the window based on the new value(s) you have entered.

The ‘Annotator’ field is empty when the program is first started, unless an *annotator* argument was provided on the command line. If you enter an annotator name (such as ‘atruth’), annotations from the specified annotation file are displayed using the same format as in the examples shown on pages 99-177 of the *MIT-BIH Arrhythmia Database Directory*, except that each normal beat is labeled using an ‘N’ rather than a bullet.

Changing the start (or end) time causes the window to be redrawn with its left (or right) edge corresponding to the selected time. *dbtool* always begins at an even multiple of one second, however. If both the start and end times are changed, only the last change is effective; use the ‘Set modes’ popup menu (see below) to change the time scale if desired.

The buttons marked ‘<<’ and ‘>>’ move backward and forward by the width of the window when selected using the left mouse button; those marked ‘<’ and ‘>’ move by half the width of the window.

If an annotator has been chosen for display, you may enter a character string in the ‘Find’ field. Each time you press RETURN (with the mouse pointer in the control panel and the ‘Find’ field selected) *dbtool* searches for the next occurrence (beyond the right edge of the window) of an annotation that matches the specified string. If *dbtool* finds a match, the display is redrawn (roughly) centered on the annotation of interest. In general, the string you enter should match what you expect *dbtool* to display: an annotation mnemonic (such as ‘V’ or ‘F’), a rhythm name (such as ‘(VT’), an integer for a signal quality change, or the text of a comment, ST change, or T-wave change annotation. The following strings can also be entered:

***s** Matches all supraventricular premature beat annotations (including APCs, NPCs, and aberrated APCs)

***v** Matches all ventricular ectopic beat annotations (including R-on-T PVCs, ventricular flutter waves, and ventricular escape beats, but not fusion PVCs)

***n** Matches most other types of beat annotations (normal, bundle branch block, atrial and nodal escape, paced, pacer fusion)

***** Matches all annotations (useful for finding the first annotation in an incompletely annotated record)

A variety of display options can be set using the popup menu that appears when ‘Set modes’ is selected. These can be used to display grids, to change scales, and to select annotation fields to be displayed.

Select the button marked ‘Log’ to record the start and end times of the current window. You will be asked to choose a name for the log file the first time you use this feature. You can also enter a brief description of the contents of the window at the time you log it; the log file is stored in *pschart* script format (see *pschart*(1)).

To exit, select the ‘Quit’ button.

The shell variable **DB** should be set and exported (see *setdb*(1)). This program runs on Sun workstations only.

BUGS

No more than one piped record (see the *ECG Database Programmer's Guide*) can be viewed in a single invocation of *dbtool*. If the signal file is a pipe, it is possible only to search forward through it (although *dbtool* caches several of the most recently displayed windows, which can be reviewed in any case).

If the display resolution does not match the value compiled into *dbtool*, the scales will be incorrect. Scales can be checked by measuring the grid if necessary.

SEE ALSO

dbplot(1), pschart(1), view(1), wave(1)

NAME

dbwhich − find a DB file and print its pathname

SYNOPSIS

dbwhich [**-r** *record*] *filename*
dbwhich [**-r** *record*] *file-type record*

DESCRIPTION

This program searches the DB path (as specified by the environment variable **DB**, see *setdb*(1)) for a specified *filename*, or for a file of a specified *file-type* (e.g., 'header' or 'atruth') which belongs to a specified *record*. If the file can be found, its full pathname is written to the standard output, and *dbwhich* exits with an exit status of zero (indicating success). If the file cannot be found, a diagnostic message, including the current value of the DB path, is written to the standard error output, and *dbwhich* exits with a non-zero exit status.

If the DB path includes '%r', use the **-r** *record* option to specify the record name to be substituted for '%r'.

SEE ALSO

setdb(1)

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 *epic*(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).

The shell variable **DB** should be set and exported (see *setdb*(1)).

FILES

dblist

This file, which should be located in one of the directories named by **DB**, 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 **DB**) 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 DB 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), epic(1), mxm(1), plotstm(1), rxr(1), setdb(1), sumstats(1)
Evaluating ECG Analyzers

NAME

epic − ANSI/AAMI-standard episode-by-episode annotation comparator

SYNOPSIS

epic -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 EC38-1994). *epic* 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.

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* Ignore episodes shorter than *time* (default: 0 seconds).

-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*.

The episode and duration statistics gathered by *epic* 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 **<ecg/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 *epic*. 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 *epic*. 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, *epic* assumes that they have not changed since they last appeared. *epic* 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, *epic* can be used repeatedly to build up line-format tables for multiple records, for further processing by *sumstats*(1).

The shell variable **DB** should be set and exported (see *setdb*(1)).

DIAGNOSTICS

non-standard comparison selected

The **-f**, **-i**, and **-t** options modify the comparison algorithms used by *epic* in ways not permitted by EC-38. 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 *epic* performs multiple passes over its input files, it cannot be used at the end of a pipe.

It's not really an epic, it's just long. By analogy to *bxb* 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), setdb(1), sumstats(1)
Evaluating ECG Analyzers (in the *ECG Database Applications Guide*)
American National Standard for Ambulatory Electrocardiographs (publication ANSI/AAMI EC38-1994); available from AAMI, 3330 Washington Blvd., Suite 400, Arlington, VA 22201 USA.

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 an 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)

AVAILABILITY

This program is part of the *psd* package, distributed with the DB Software Package, version 9.3 and later.

NAME

fir − general-purpose FIR filter for DB 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 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.

-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.

The shell variable **DB** should be set and exported (see *setdb*(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)

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* 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).

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

SEE ALSO

fft(1), ihr(1), lomb(1), memse(1), plot2d(1), tach(1)

AVAILABILITY

These programs are part of the *psd* package, distributed with the DB Software Package, version 9.3 and later.

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). Any intervals for which the calculated heart rate would differ by more than the specified tolerance are simply excluded from the output series.

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

-h
: Print a usage summary.

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

-x
: Exclude all intervals adjacent to abnormal beats. By default, such intervals are included (unless they are rejected by the *tolerance* criterion), and a third column is used to flag these intervals as abnormal (a zero in the third column means the interval is normal, a one means it is abnormal).

Reference (‘atruth’) 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).

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

lomb(1), setdb(1), sqrs(1), tach(1)

AVAILABILITY

This program is included in the DB Software Package, version 9.3 and later.

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.

AVAILABILITY

This program is part of the *psd* package, distributed with the DB Software Package, version 9.3 and later.

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

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.

-s Smooth the output.

The original version of this program was based on the algorithm described in Press, W.H, and Rybicki, G.B., ‘‘Fast algorithm for spectral analysis of unevenly sampled data’’, *Astrophysical J.* 338:277-280 (1989). It has been rewritten using the version of this algorithm presented in 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).

Among many other applications, this program can be used to estimate heart rate power spectra, in combination with *ihr* (see Moody, G.B., ‘‘Spectral analysis of heart rate without resampling’’, *Computers in Cardiology 1993*, pp. 715-718 (IEEE Computer Society Press, 1993)). 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)

AVAILABILITY

This program is part of the *psd* package, distributed with the DB Software Package, version 9.3 and later.

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:

-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.

-l *n*
: Handle input series up to *n* points in length. Default: *n* = 16384, although you should be cautious about using *memse* with such long input series.

-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.

-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.

SEE ALSO

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

AVAILABILITY

This program is part of the *psd* package, distributed with the DB Software Package, version 9.3 and later.

NAME

mfilt − general-purpose median filter for DB 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.

-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.

The shell variable **DB** should be set and exported (see *setdb*(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)

AVAILABILITY

This program is included in version 8.1 and later versions of the DB Software Package.

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 *dbcollate*(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.

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

bxb(1), dbcollate(1), setdb(1)

AVAILABILITY

This program is included in version 9.3 and later versions of the DB Software Package.

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 EC38-1994, 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.

SEE ALSO

bxb(1), ecgeval(1), epic(1), rxr(1), setdb(1), sumstats(1)
Evaluating ECG Analyzers (in the *ECG Database Applications Guide*)
American National Standard for Ambulatory Electrocardiographs (publication ANSI/AAMI EC38-1994); available from AAMI, 3330 Washington Blvd., Suite 400, Arlington, VA 22201 USA.

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 DB 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, *atruth* 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:

$SNR = 10 \log (S/Na^2)$

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 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.

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

(*) On the first edition of the CD-ROM, signal 1 was omitted from record 118_10. To reconstruct record 118_10 as intended, compile *mk118_10.c* (in the *convert* directory of the DB software package) and run it. The second edition of the CD-ROM contains the corrected version of this record.

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), *epic*(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

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

bxb(1), epic(1), mxm(1), rdann(1), rxr(1), setdb(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).

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 portable, quick-and-dirty replacement for Paul Albrecht's elegant but unsupported *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 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)

gnuplot is freely available in C source form by anonymous FTP from *prep.ai.mit.edu* and many other sites.

AVAILABILITY

This program is included in version 9.3 and later versions of the DB Software Package.

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 *epic*(1) using its **-S**, **-S0**, or **-S1** option, and generates a PostScript page description for a scatter plot of these data, as specified by a draft AAMI standard for ambulatory ECGs. The standard output of *plotstm* may be printed directly on any PostScript device.

SEE ALSO

ecgeval(1), epic(1)
Evaluating ECG Analyzers

NAME

pschart − produce annotated ‘chart recordings’ on a PostScript device

SYNOPSIS

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

DESCRIPTION

pschart produces high-quality annotated plots of ECG database 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 laser printer while producing visually pleasing results. If the output is destined for a 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: ‘atruth’). 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 ""**’.

-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 under each strip (default: no grid).

-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 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, 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 *dbcal*(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).

-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 *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). The environment variables **DB** and **DBCAL** should be set and exported (see *setdb*(1)).

FILES

/usr/local/lib/ps/pschart.pro
default PostScript prolog file.

BUGS

High quality takes time. 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. A SPARCprinter attached to a Sparc IPC or equivalent can render *pschart* output at nearly full speed (about 28 seconds for the first page, 8 seconds for each subsequent page).

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.

SEE ALSO

dbplot(1), dbtool(1), psfd(1), setdb(1), view(1), wave(1), xform(1)

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 ECG database 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 laser printer while producing visually pleasing results. If the output is destined for a 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: ‘atruth’). 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 Print counter values after time stamps in the left margin.

-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.

-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, 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 *dbcal*(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).

-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). The environment variables **DB** and **DBCAL** should be set and exported (see *setdb*(1)).

FILES

/usr/local/lib/ps/psfd.pro
default PostScript prolog file.

BUGS

High quality takes time. A full page, with grids and default scales, typically takes about 3–4 minutes to render on an Apple LaserWriter, or about 6–8 minutes on a Linotronic 1200 dpi phototypesetter. A SPARCprinter attached to a Sparc IPC or equivalent can render *psfd* output at nearly full speed (about 28 seconds for the first page, 8 seconds for each subsequent page).

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

dbplot(1), dbtool(1), pschart(1), setdb(1), wave(1)

NAME

rdann − read a DB annotation file

SYNOPSIS

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

DESCRIPTION

rdann reads the annotation file specified by *record* and *annotator*, and writes an ASCII 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:

-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.

-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).

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

-x Use an alternate time format for output (the first two columns are the elapsed times in seconds and in minutes, replacing the *hh:mm:ss* and sample number columns in the default output).

The **-f** and **-t** options may be used to select a portion of an annotation file for printing. Their arguments are usually given as times in *hh:mm:ss* format; see the description of *strtim* in the *ECG Database Programmer's Guide*, as well as the comments below, for other formats.

Beat numbers beginning with 0 are implicitly assigned by *rdann* to each QRS annotation in an annotation file. If the argument of the **-f** option begins with ‘#’, it is taken to be the beat number of the first QRS annotation to be printed (any non-QRS annotations that immediately precede this annotation are also 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.

The shell variable **DB** should be set and exported (see *setdb*(1)).

Example

rdann -a atruth -r 200 -f 0 -t 5:0 -p V

This command prints on the standard output all “V” (PVC) annotations in the first five minutes of the *atruth* (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.

SEE ALSO

setdb(1), wrann(1)

NAME

rdsamp – read DB 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. Each line of output contains the sample number and samples from each signal, beginning with channel 0, separated by tabs.

Options include:

-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).

-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.

-s *signal-list*
Print only the signals named in the *signal-list* (one or more input signal numbers, separated by spaces; default: print all signals). This option may be used to re-order or duplicate signals.

-t *time* Stop at the specified *time*. By default, *rdsamp* stops at the end of the record.

-v Print column headings.

The shell variable **DB** should be set and exported (see *setdb*(1)).

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 *rdsamp* 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 *rdsamp* for further information.

SEE ALSO

rdann(1), setdb(1), wrsamp(1)

NAME

rxr – ANSI/AAMI-standard run-by-run annotation comparator

SYNOPSIS

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

DESCRIPTION

This program implements the run-by-run comparison algorithm described in *Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms*, an AAMI Recommended Practice (AAMI ECAR-1987). Using options **-C**, **-L**, or **-S**, *rxr* implements the run-by-run comparison algorithm specified by the current American National Standard for ambulatory ECG analyzers (ANSI/AAMI EC38-1994). *rxr* is the reference implementation of these algorithms, and must be used to obtain the run-by-run performance statistics cited in EC38 in order to be in compliance with the standard. The EC38 comparison algorithm includes all of the ECAR statistics (so that both sets of requirements are satisfied by using the EC38 algorithm), with additional statistics on detection of runs of supraventricular ectopic beats (SVE runs).

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 EC-38 and in ECAR).

Options include:

-c *file* Append condensed reports (ECAR Table 10 format) 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 (ECAR Table 13 format) to *file* (see below).

-L *file file2*
As for **-l**, but report SVE run statistics in *file2*.

-s *file* Append standard reports (ECAR Table 3 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.

The shell variable **DB** should be set and exported (see *setdb*(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), epic(1), mxm(1), setdb(1), sumstats(1)

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

Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms (publication AAMI ECAR-1987)

American National Standard for Ambulatory Electrocardiographs (publication ANSI/AAMI EC38-1994)

The last two publications are available from AAMI, 3330 Washington Blvd., Suite 400, Arlington, VA 22201 USA.

NAME

sample − digitize and replay analog signals (MS-DOS only)

SYNOPSIS

sample [*options* ...]]

DESCRIPTION

sample generates database records from analog signals, and generates analog signals from database records.

When used to digitize analog signals, the outputs of *sample* are one or more signal files and a header file. The record name, the number of input signals, the sampling frequency, the duration of the record, and the storage format may be specified interactively or by command-line options and configuration files.

When used to play back database records, the outputs of *sample* are one or more analog signals.

Options include:

-b Batch mode (do not prompt to begin digitization or playback).

-d *type* Set up for use with the specified device. *Type* must be one of '1200/2', '1200/2B', '1200/3', '1200/3B', '1200/4', '1200/4B', '2400/4', '2400/5', or '2400/6' (corresponding to Microstar DAP model numbers). Use a *type* ending with 'B' only if you have a DAP 1200 series board configured for bipolar inputs. Use the option **-d help** to get a list of supported device types. If no **-d** option is given, *sample* uses the DAPL 'hello' command to determine the device type automatically.

-f *time* Begin playback at the specified *time* (effective only in combination with the **-i** option).

-h Print on-line help (a brief description of options).

-i *record*
Play back the specified *record* (may not be used with the **-n**, **-N**, or **-o** options). Unless an analog output expansion board has been installed, and *sample* has been compiled with a suitably increased value of *DAPOUTPUTS*, or a DAPL program embedded in the header file for *record* (see below) specifies otherwise, playback is restricted to signals 0 and 1 only. Except when using 'raw' format, the analog outputs are determined by the 12 least significant bits of each sample. (*sample* assumes that any record whose header file specifies format 16, ADC resolution 16, and all signals in the same signal file, is a 'raw' format record.)

-n *record*
Create the specified *record* (both signal and header files).

-N *record*
Create a header file for the specified *record*, but do not digitize.

-o *record*
Digitize, using signal specifications in the existing header file for *record*.

-p *file* Use the DAPL program in the specified file.

-s *n* Record or play back 1 of every *n* samples in each signal (not effective with 'raw' format I/O, see below).

-t *time* Stop digitization or playback at the specified *time*.

-x *n* Digitize or play back at *n* times real time ($n > 0$; default: 1). This option is not effective with **-p**, or when the header file specified by **-i** or **-o** contains a usable DAPL program (see below).

-1 Play back only one signal (signal 0; effective only with **-i**).

The shell variable **DB** should be set and exported (see *setdb*(1)).

HARDWARE REQUIREMENTS

sample runs under MS-DOS on an ISA-bus (AT-bus) system that contains a Microstar DAP 1200 or 2400 series board (available from Microstar Laboratories, 2265 116th Avenue NE, Bellevue, WA 98004 USA; telephone +1 206 453 2345). All DAP 1200 and 2400 series boards listed below include 12-bit analog-to-digital and digital-to-analog converters, analog multiplexers, programmable gain amplifiers, timers, and an

Intel 80186 CPU; DAP 2400 series boards include a Motorola 56001 DSP and provisions for external triggering. All models have 8 differential (16 single-ended) analog inputs and 2 analog outputs, and 16 bits each of digital inputs and outputs. With expansion boards, a single DAP can have up to 256 differential or 512 single-ended analog inputs, 66 analog outputs, and 128 bits of digital input and output; it is also possible to use up to seven DAP boards in a single system.

The range of sampling frequencies available depends on the model. DAP 1200/2 and 1200/3 boards can digitize at aggregate rates of 31.25 to 50,000 Hz; for DAP 1200/4 and 2400/4 boards, the range is 38.46 to 156,000 Hz, and for DAP 2400/5 and 2400/6 boards, the range is 62.5 to 235,000 Hz. Although several hundred thousand samples can be acquired at the maximum input sampling frequency, the maximum sustained throughput to disk is considerably lower than the maximum sampling frequency specifications imply; using a 12 MHz 286-based system, a DAP 2400/5, a disk controller with 256K of cache, and a 16 msec disk, for example, the maximum sustained throughput is about 25 KHz (100 KHz using ‘raw’ format, see below). The minimum sampling frequencies listed above refer only to those obtainable using the on-board timer and the DAPL program generated by *sample* (see below); if necessary, slower sampling can be achieved using an external clock with DAP 2400 boards, or by writing a special-purpose DAPL program.

The DAP 2400 is much better suited for playback than is the DAP 1200. Each of the two analog outputs of the DAP 1200 can accept samples at 1, 2, 5, 10, 25, 50, 125, or 250 Hz, but not at any other rates. Note that *sample* generates DAPL code for playback on DAP 2400 boards only, although DAP 1200 playback code can be provided by the user at run time if necessary (see below). *sample* can generate analog signals using a DAP 2400 at sustained rates up to about 3 KHz; much faster output (up to 250 KHz with a DAP 2400/6) is possible for short intervals (up to the amount that can be loaded into the DAP’s local memory).

Microstar Laboratories produces DAP boards that do not include analog-to-digital or digital-to-analog converters; *sample* is not intended for use with these boards, which have model designations ending in ‘D’. DAP boards with protected analog inputs have model designations ending in ‘P’; these boards have slightly lower maximum sampling rates than the corresponding standard models.

Successful digitization always requires that the analog inputs be band-limited so that there is essentially no power at frequencies in excess of half of the sampling frequency. In almost all cases, this means that you must use properly designed analog antialiasing filters between the signal source and the DAP inputs. Before assuming that your inputs are band-limited because of known characteristics of the signals, consider carefully if your *noise* is also band-limited – it’s probably not!

During playback, you may wish to use similar low-pass filters to reduce the ‘staircase’ effect characteristic of signals generated by D/A converters.

sample has been tested using DAPL version 3.3; the digitization and single-channel playback functions have also been tested using DAPL version 3.2 (note that the multi-channel playback code generated by *sample* uses the ‘separate’ command, which is new in DAPL 3.3).

PERFORMANCE HINTS

sample uses the DB library functions *getvec* and *putvec* to read and write signal files. When digitizing, the advantages of using *putvec* are that it permits the use of a variety of standard, compact storage formats, and that it calculates signal checksums that are used subsequently by other DB applications to verify the integrity of the signal files. During playback, the use of *getvec* permits *sample* to read any existing DB record without the need to reformat it. When digitizing or playing back ‘raw’ format records (compatible with Microstar’s DAPview or DLOG binary files), *getvec* and *putvec* are bypassed with low-level *read* and *write* calls for maximum speed. A side effect of using ‘raw’ format in digitization is that the generated header files for newly created records lack sample counts and checksums, since these are produced only by *putvec*. ‘Raw’ format files can be converted to any standard (‘cooked’?) DB signal file format using *xform*, which will generate sample counts and checksums. When using *xform* on ‘raw’ format files, it is advantageous to set its output signal gains to 1/16 of those specified in the header file for the ‘raw’ format record, in order to obtain right-justified 12-bit samples (DAP ‘raw’ format consists of 12-bit samples left-justified in 16-bit words). If both *sample* and *xform* are allowed to use their default gain settings, this scaling is performed automatically by *xform*.

COMPILATION

Please read this *entire* section before trying to compile *sample*!

To compile *sample* successfully, you will need Microsoft C, version 5.0 or later, or Turbo C/C++, and four files supplied by Microstar with all versions of the DAP 1200 and 2400 (Microstar's Advanced Development Toolkit is *not* required in order to compile or use this program). These files are *c_lib.c*, *clock.h*, and *ioutil.h*, which should be installed in a directory searched by your C compiler for *#include <...>* files (for Microsoft C, one of the directories specified by the *INCLUDE* environment variable); and *cdapl.lib*, which should be installed in a directory searched by your C compiler's link utility for libraries (for Microsoft C, one of the directories specified by the *LIB* environment variable). If you are using Microsoft C, copy *cdapl5.lib* from the Microstar distribution diskettes into your library directory and rename it *cdapl.lib*.

If you have not already done so, you must also compile and install a large memory model version of the DB library (see *makefile.dos* in the *lib* directory). Although the program compiles and links successfully using the small memory model and Microstar's *cdaps5.lib*, there is apparently a bug in *cdaps5.lib* that wedges the DAP (a condition that can be remedied only by power-cycling the system).

If you have installed one or more analog expansion boards, modify the values of *DAPINPUTS* and *DAPOUTPUTS* as appropriate (see *sample.c*). Note that these constants specify the number of available inputs and outputs, *not* the number of signals that can be digitized or played back at once using *sample* (determined by *MAXSIG*, defined in *<ecg/db.h>*). If you need to digitize a larger number of signals than is permitted by the current value of *MAXSIG*, modify the definition of *MAXSIG* in the copy of *db.h* that is in the *lib* directory, and recompile the DB library (as well as any applications that you wish to use with records created by *sample*).

Efficiency, hence maximum output rate, is improved by using large I/O buffer sizes. By default, *sample* uses 10K-byte buffers; depending on your system's available memory and your disk controller, you may be able to improve your throughput by adjusting this value. Do so by redefining the symbol *DBUFSIZE*. Note that the default buffer size is overridden at run time by explicit (non-zero) block size specifications in the header file, if present (see *header*(5)).

This program has been successfully compiled and tested using Microsoft C 5.1, 6.0, and 7.0, with the command

cl -Ox -Ml sample.c -link dbl cdapl

and using Turbo C 2.01, and Turbo C++ 1.01, with the command

tcc -O -A -ml sample.c dbl.lib cdapl.lib

Note that you may need to use **-I** and **-L** options to instruct your compiler where to find *#include* files and libraries; see your compiler manual for further information. The symbols *DAPINPUTS*, *DAPOUTPUTS*, and *DBUFSIZE* may be redefined using command-line **-D** options with any of these compilers. If you compile using Microsoft C/C++ 7.0 (or later) with the **-Za** (ANSI C) option, add **oldnames** as a final command-line argument, to link a library containing aliases for functions such as *open()*, which have been renamed in the standard MSC 7.0 library to conform with ANSI namespace rules.

If you tell *install* (the MS-DOS installation program for the DB software package) to compile *sample* using Microsoft C, *install* looks for *cdapl5.lib*, and uses it in preference to *cdapl.lib* if both are present (thus you can have DAP libraries for both Microsoft C and Turbo C installed at the same time). *install* also detects the presence of *oldnames.lib* and adds it to the command line if necessary.

DAPL PROGRAMS

DAP boards are controlled by programs written in Microstar's DAPL (DAP language), which are downloaded to the on-board 80186 for execution. *sample* follows this procedure for obtaining the DAPL program to be used:

1. If you use the **-p** option to specify the name of a DAPL file, *sample* downloads the contents of that file. The first line of the file should contain a DAPL comment line of the form '*; start a,b*', which specifies the DAPL command required to begin execution of the program once it has been downloaded. If this line is missing, *sample* uses the command '*start a*'. Unless the DAPL file is unreadable or empty, *sample* skips steps 2 and 3, and the **-x** option has no effect.

2. If you use the **-i** or **-o** options to specify the record name of an existing header file, *sample* will read info strings from the header file. If *-i* has been specified, and any of the info strings begin with ‘<’, the initial ‘<’ character in each such string is discarded, and the remainder is treated in the same manner as for a DAPL program specified using **-p**. Similarly, if **-o** has been specified, *sample* compiles the DAPL program from any info strings beginning with ‘>’. If *sample* finds any DAPL code in this step, it does not execute step 3, and the **-x** option has no effect.

3. If *sample* has not obtained a DAPL program in steps 1 or 2, it generates one based on the signal specifications obtained from the header file read in step 2, if any, or interactively from the user. If specifications for digitization are obtained from a header file without an embedded DAPL program, *sample* assigns differential inputs D0, D1, ... to signals 0, 1, ..., provided that the number of signals does not exceed *DAPINPUTS*/2; otherwise, *sample* assigns single-ended inputs S0, S1, ... to signals 0, 1, ... Digital inputs (input ‘B’) may be specified in a DAPL program or interactively. In playback mode, *sample* generates DAP 2400-compatible code only; to play back a record using a DAP 1200, you must supply DAPL code either in the header file or using **-p**.

When *sample* is used to create a record, it writes a copy of the DAPL program that it uses into the newly-created header file, as a set of info strings formatted as described in step 2 above. By using the **-N** option, a DAPL program may be written into a prototype header file without actually digitizing any signals. This is an easy way to bootstrap the design of a custom DAPL program for use with *sample*. Custom DAPL programs may be used for applications that perform operations such as digital filtering on-the-fly, or for low frequency sampling.

An example of a header file with a moderately complex embedded DAPL program is *sample8.hea*, in the *app* directory of the DB software package.

DISCLAIMER

Please note that the author has no connection with Microstar Laboratories other than as a satisfied customer; specifically, neither MIT nor the author assumes any responsibility whatsoever for the performance of Microstar hardware or software.

BUGS

If no **-d** option is given, *sample* displays the diagnostic message “Testing DAP ...” briefly before printing the DAP type. If *sample* cannot establish communications with the DAP, however, it will stop after displaying the “Testing DAP ...” message; in this case, it is usually necessary to power-cycle the DAP (a system reset is usually not sufficient) in order to restore communications. This behavior of *sample* is not a bug in itself; the DAP board can be put into this uncommunicative (i.e., wedged) state by *sample* and other programs, however. The most likely reasons for the DAP to become wedged are attempting to use sampling frequencies that are too high (especially for playback) or improperly constructed custom DAPL programs in header files. It is also possible to run DAPL programs that take a very long time to execute; in this case, it may be difficult to know if the DAP is running or wedged.

SEE ALSO

setdb(1), xform(1), header(5)

NAME

setdb, cshsetdb, dossetdb – set DB environment variables

SYNOPSIS

. setdb
source cshsetdb
dossetdb

DESCRIPTION

sh, *bash*, and *ksh* shell users:

setdb sets the shell variables **DB** and **DBCAL** to the appropriate values to permit access to on-line ECG databases. It must be executed using the ‘‘.’’ as shown above. A frequent user of the databases will find it convenient to include an invocation of *setdb* in the *.profile* file.

csh and *tcsh* users:

cshsetdb sets **DB** and **DBCAL** similarly for the C-shell. It must be executed using ‘‘source’’ as shown above. It may be convenient to include this command in the *.cshrc* file.

MS-DOS users:

dossetdb sets **DB** and **DBCAL** similarly for MS-DOS; it also sets **VIEWP** (see *view*(1)). It may be convenient to include this command in the *autoexec.bat* file (under MS-DOS 3.3 and later, include the command ‘‘call dossetdb’’ within *autoexec.bat*; under earlier versions of MS-DOS, use ‘‘command /c dossetdb’’, or append the contents of *dossetdb.bat* to *autoexec.bat*).

These programs are shell scripts that must be customized before using them for the first time on a new machine.

ENVIRONMENT

DB 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 *db*(3) library search for their database input files in the order specified by **DB**. If **DB** is not set, searches are limited to the current directory. Under UNIX, directory names are separated by colons (:), and the format of **DB** 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 **DB** is that of the MS-DOS **PATH** variable (colons may be used following drive specifiers within **DB** in this case). The Macintosh does not support environment variables as such; on the Macintosh, the value of **DB** is defined in *dblib.h* (one of the DB library source files, not usually included in DB applications), 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. Under any environment, if the value of **DB** begins with ‘@’, the remainder of the string is taken as the name of an ‘‘indirect DB path file’’ that defines the database path in the format described above. This feature was introduced in DB library version 8.0, mainly to permit Macintosh users to modify the DB path without recompiling the DB library, but it is also useful under MS-DOS to avoid the 128-character limit on the length of environment variables. Indirect DB path files can be nested up to 10 levels deep.

DBCAL

The name of the DB calibration file (see *dbcal*(5)). The usual rules for finding DB files by searching the DB path apply to the DB calibration file, so the value of **DBCAL** need not be an absolute path name. The DB calibration file is used by DB applications that need to plot signals at standard scales, as well as by *calibrate*(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 **DBCAL** is not set by the user, the DB library uses a default DB calibration file (*dbcal*, named in *dblib.h*). If the DB calibration file is not readable, programs that rely on it may not choose appropriate scales for some types of signals.

NAME

sigamp − measure signal amplitudes of a DB record

SYNOPSIS

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

DESCRIPTION

sigamp measures either baseline-corrected RMS amplitudes or normal QRS peak-to-peak amplitudes for all signals of the specified *record*. It makes up to 250 measurements 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.

-p Print results in physical units (default: ADC units).

-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).

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

calibrate(1)

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’ could be omitted from the command.)

Applications built using the DB library (version 9.2 or later) are able to correct for skew (the skew correction is performed by the DB 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 DB 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 DB library.

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

xform(1), header(5)

AVAILABILITY

This program is included in the DB Software Package, version 9.0 and later.

NAME

snip − copy an excerpt of a DB 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.

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

The shell variable **DB** should be set and exported (see *setdb*(1)).

FILES

new-record.dat	output signal file
header.*new-record*	output header file (under MS-DOS, *new-record*.hea)
annotator.new-record	output annotation file (under MS-DOS, *new-record.ann*)

SEE ALSO

xform(1), setdb(1)

NAME

sortann − rearrange annotations in canonical order

SYNOPSIS

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

DESCRIPTION

Applications that use the DB 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 *chan* attributes).

sortann rewrites the annotation file specified by *record* and *annotator*, arranging its contents in canonical (*time* and *chan*) order. By default, DB applications run *sortann* as needed (from within *dbquit* or *oannclose*). If the environment variable **DBNOSORT** 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 DB application.

If the input contains two or more annotations with the same *time* 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).

Running out of memory is unlikely unless:

1. you have less than 2 Mb of memory and you are attempting to sort a 24-hour or longer annotation file.
2. you have compiled *sortann* using a 16-bit compiler and you are attempting to sort more than about 4000 annotations (note that the precompiled versions of *sortann* for MS-DOS and UNIX do not have this limitation).

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.

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

mrgann(1), setdb(1)

NAME

sqrs − 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 *ECG Database 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). For best results on adult human ECGs, use *xform* to resample the input signal at 250 Hz if a different sampling frequency was used originally (or use *sqrs125*, a variant of *sqrs* designed for signals sampled at 125 Hz). For other ECGs, it may be necessary to experiment with the input sampling frequency and the time constants indicated in the source file.

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*).

-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* to be used for QRS detection (default: 0).

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

The shell variable **DB** should be set and exported (see *setdb*(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 atruth qrs

SEE ALSO

bxb(1), rdann(1), setdb(1), xform(1)

NAME

sumann − summarize the contents of a DB 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.

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

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

rdann(1), setdb(1)

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 4.6.1 and 5.6 of *Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms* (AAMI ECAR-1987), and additional statistics specified by the *American National Standard for Ambulatory Electrocardiographs* (ANSI/AAMI EC38-1994).

To use this program, first generate a line-format report *file* using the **-l** or **-L** options of *bxb*(1), *epic*(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), epic(1), mxm(1), plotstm(1), rxr(1)
Evaluating ECG Analyzers (in the *ECG Database Applications Guide*)
Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms (publication AAMI ECAR-1987)
American National Standard for Ambulatory Electrocardiographs (publication ANSI/AAMI EC38-1994)
The last two publications are available from AAMI, 3330 Washington Blvd., Suite 400, Arlington, VA 22201 USA.

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 2*k* 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 DB 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 ('atruth') 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).

The shell variable **DB** should be set and exported (see *setdb*(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

setdb(1), sqrs(1)

NAME

view, vsetup – DB browser for MS-DOS

SYNOPSIS

view
view *record*
view *record annotator*
view *record annotator time*
view *record annotator type*
vsetup
vsetup -s *file*
vsetup *mode*

DESCRIPTION

This program reads one or more signal files and an annotation file (if one is available), and displays the signals with annotations at the standard chart recorder scale of 25 mm/sec and 10 mm/mV on systems equipped with a CGA, Hercules, EGA, VGA, or compatible graphics display adapter. Run *vsetup* to determine a suitable display mode and scaling constants before using *view* for the first time (see *vsetup* below).

If no *record* is specified on the command line, *view* asks for a record name when it starts. If no *annotator* is specified on the command line, *view* attempts to find an annotation file named *record***.atr**; otherwise, it attempts to find one named *record***.***annotator*. If the annotation file does not exist or cannot be read, *view* simply displays the signals. A *time* argument, if present, causes *view* to display its first screenful such that the left-hand edge of the screen corresponds to the specified time. A *type* argument, if present, causes *view* to search for the first occurrence of the specified annotation type in the annotation file, and to center the first screenful on that annotation if possible.

view is an interactive program. Once the first screenful has been displayed, *view* waits for additional commands, which can be typed in the prompt area near the bottom of the screen, and which are executed when you press ENTER. These commands can be in any of the formats permissible for *time* or *type* command-line arguments; such commands have the same effect as if given on the command line. You may also merely press ENTER to see the next screenful; if the previous action involved searching for an annotation, this will cause *view* to search for the next occurrence of the same annotation type that does not already appear on the screen. The **g** command may be used to turn the grid display on or off; initially, the grid display is off. When the grid is visible, it marks 0.2 second and 0.5 millivolt intervals. The **m** command may be used to turn annotation marker bars (dotted lines above and below each annotation showing the exact location of the annotation fiducial mark) on or off; initially, annotation marker bars are on. Finally, use the **x** command to exit from *view* and return to the DOS prompt.

Current versions of *view* (4.0 and later) support the use of a mouse to navigate through the record. If a mouse is available, *view* displays buttons at the bottom of the screen; click left on these to move forward or backward by whole or half screenfuls. If the mouse pointer is above the level of the buttons, clicking left or right moves the pointer left or right one annotation at a time, recentering the display if the pointer would otherwise move out of the display region.

vsetup is used to calibrate your monitor for *view*. If *vsetup* is able to identify a usable graphics mode, it will ask you to measure the dimensions of a rectangle that it draws on your screen. When it finishes, it will print instructions for setting an environment variable that is required by *view*, unless you have used the **-s** option; in the latter case, the MS-DOS command needed to set the variable is appended to the specified *file*. If your graphics adapter is capable of several display modes, and you prefer to use a different mode from that selected by *vsetup*, you can run *vsetup* with an integer argument that corresponds to your preferred *mode*, from among the following:

Argument	*Display mode*
4	320 x 200, 4 colors
5	320 x 200, 4 grey levels
6	640 x 200, black and white
8	720 x 348, black and white [1]

13	320 x 200, 16 colors
14	640 x 200, 16 colors
15	640 x 350, black and white
16	640 x 350, 4 or 16 colors
17	640 x 480, black and white
18	640 x 480, 16 colors
19	320 x 200, 256 colors [2]
258	800 x 600, 16 colors [3]
259	800 x 600, 256 colors [2,3]
260	1024 x 768, 16 colors [3]
261	1024 x 768, 256 colors [2,3]
262	1280 x 1024, 16 colors [3]
263	1280 x 1024, 256 colors [2,3]

[1] Mode 8 can be used only with a Hercules Graphics Card, Graphics Card Plus, Incolor Card, or a compatible. To use mode 8 with one of these cards, load *msherc.com* (from the *bin* directory of your CD-ROM or software distribution diskette, or from your Microsoft C, Pascal, or Fortran compiler diskettes) before using *vsetup* or *view*. Do so by typing *msherc* from within the directory where you find *msherc.com* (you may wish to include this step in your *autoexec.bat*). If you have both a Hercules monochrome card and a color video card in the same system, type *msherc /h* instead.

[2] *view* uses at most 5 colors, so the 256-color modes offer no advantage over the 16-color modes. In the 16- and 256-color modes, the background is white; in the 4-color and black-and-white modes, the background is black.

[3] These modes can be used only with a VESA-compatible SVGA and a compatible monitor. *vsetup* will not identify these modes automatically, because of the potential for damage to your monitor if your SVGA card supports these modes but your monitor does not. **Do not attempt to use these modes unless your monitor supports them. Otherwise, you risk damaging your monitor.**

ENVIRONMENT

DB 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 *db*(3) library search for their database input files in the order specified by **DB**. If **DB** is not set, searches are limited to the current directory. Under MS-DOS, directory names are separated by semicolons (;), and the format of **DB** is that of the MS-DOS **PATH** variable (colons may be used following drive specifiers within **DB** in this case).

DBCAL

The name of the DB calibration file (see *dbcal*(5)), which must be in a directory named by **DB** (see immediately above). This file is used by *view* to determine standard scales for signals other than ECGs. If **DBCAL** is not set, or if the file named by **DBCAL** is not readable, these signals may be drawn at incorrect scales.

VIEWF

The pathname for the font file containing characters to be drawn by *view*. **VIEWF** should name a Microsoft Windows-compatible '*.fon*' file. By default, if **VIEWF** is not set, *view* checks the **PATH** variable to see if Microsoft Windows is available, and uses *modern.fon* in the Windows *system* subdirectory if so, or a file of the same name in the current directory if not; otherwise, *view* uses text mode to draw characters. Note that text mode character placement is constrained to text mode character cells; annotations shown using text mode will not be placed as precisely as those shown using graphics mode.

VIEWP

Instructions for setting this variable are printed by *vsetup*. The format for the command to do so is

set VIEWP=*mode,left,right,top,bottom,width,height*

where *mode* is one of those specified in the table above, *left*, *right*, *top*, and *bottom* are the pixel column and row numbers corresponding to the edges of the display area to be used by *view*, and *width* and *height* are the dimensions of the display area in millimeters.

After determining appropriate values for these variables, you may wish to add commands for setting them to your *autoexec.bat* file. The standard installation procedure determines appropriate values interactively and inserts the necessary commands into *dossetdb.bat* (see *setdb*(1)).

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 *view* that do not support color or greyscale output, SVGA display modes, **VIEWF**, annotation marker bar display, or mouse interaction.

The second editions of the MIT-BIH Arrhythmia Database CD-ROM and of the MIT-BIH Polysomnographic Database CD-ROM contain a version of *view* that supports all of these features. This version, however, requires that the record name be supplied on the command line. (As described above, earlier and later versions of *view* obtain the record name interactively if it is not supplied on the command line.)

SEE ALSO

dbplot(1) (for UNIX), dbtool(1) (for SunView), pschart(1) (for PostScript), wave(1) (for X11), wview(1) (for MS Windows)

NAME

wave − waveform analyzer, viewer, and editor for the X Window System

SYNOPSIS

wave -r *record*[**+***record* ...] [*options* ...]

DESCRIPTION

wave can be used to view the specified DB *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*. (One way to do this is by pointing your Web browser to **file:///usr/local/help/html/wug/wug.htm**, or to **http://ecg.mit.edu** for the latest version.) If the *Guide* is unavailable, print a copy of the on-line manual; run *wave* without any command-line arguments to get instructions for doing so. Note that this operation does not require an X display.

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 the on-line manual for notes on annotation editing.

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* [*signal* ...]
: 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, 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). Use *rlogin*, *telnet*, or a similar method to log into the host system from the local system, and 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 not set, *wave* will not be able to open windows on your display unless your display is attached to the host system directly (not via network connection).

DB The database path (see *setdb*(1)). If not set, *wave* can find database files only in the current directory. 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.

DBCAL
The DB calibration file (see *setdb*(1) and *dbcal*(5)). If not set, *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*.

PRINTER
The name of a PostScript 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 Quercia and O'Reilly's 'X Window System User's Guide'. 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/ecg/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 (0: 50 mm/min; 1: 125 mm/min; 2: 250 mm/min; 3: 500 mm/min; 4: 12.5 mm/sec; 5: 25 mm/sec (default); 6: 50 mm/sec; 7: 125 mm/sec; 8: 250 mm/sec).

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 3 (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).

$DB The DB path (from the Load window).

$DBCAL
The name of the DB 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	calibrate -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 Solaris 2.x and Linux versions of the XView library do not have this bug.

On some 24-bit displays, an X server bug causes *wave* to start with an empty signal window. Using any of the navigation controls, or resizing the window, should make the signals visible. 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 *ECG Database 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.

SEE ALSO

dbplot(1), dbtool(1), pschart(1), view(1), wview(1)
xview(7) (in Sun's *DeskSet Environment Reference Guide*, or on-line; on some systems, this man page is known as *xview*(1))
WAVE User's Guide

AVAILABILITY

Versions of *wave* for use on Sun SPARCstations and clones (under SunOS or Solaris), and on Intel 386-compatible PCs (under Linux) are included on the third edition of the *MIT-BIH Arrhythmia Database CD-ROM*, and on the *Software for Physiologic Databases with Samples CD-ROM*. Demonstration versions of *wave* (fully functional except that saving the results of annotation editing operations is not possible) may be obtained from **http://ecg.mit.edu** (or by anonymous FTP from **ecg.mit.edu**) and on the *Samples of Physiologic Databases CD-ROM*.

wave is not part of the DB Software Package, but is intended to work with this package on systems that 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 supplied on our CD-ROMs that include *wave*, and are also available from **sunsite.unc.edu**, **tsx-11.mit.edu**, 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).

The second edition of the *MIT-BIH Arrhythmia Database CD-ROM* contained an earlier version of *wave* (for Sparc SunOS only) that lacked many of the features described here. Refer to the documentation included on that CD-ROM for details.

NAME

wrann – write a DB 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). 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*.

The shell variable **DB** should be set and exported (see *setdb*(1)).

SEE ALSO

rdann(1), setdb(1)

NAME

wrsamp − write DB signal files

SYNOPSIS

wrsamp [*options* ...] *column* [*column* ...]

DESCRIPTION

wrsamp reads text-format input and writes the specified *columns* in DB signal file format 16 (see *signal*(5)), either to the standard output or to a disk file (see the **-o** option below).

Normally, *wrsamp*'s input is line- and column-oriented, with *line separator characters* (usually ASCII line-feeds) separating input lines, and *field separator characters* (usually spaces or tabs) 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.

Lines are identified by line number. The first line of input 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 missing or improperly formatted, *wrsamp* produces a warning message identifying the line and column numbers, and inserts a zero in the output in place of the missing or improperly formatted sample.

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.)

-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). 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 in format 16 (see *signal*(5)), and does not create a header file.

-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.

The shell variable **DB** should be set and exported (see *setdb*(1)).

Example

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*.

SEE ALSO

rdsamp(1), setdb(1), xform(1), signal(5)

AVAILABILITY

This program is included in version 8.1 and later versions of the DB Software Package.

NAME

wview − DB browser for MS Windows

SYNOPSIS

wview
wview *record*
wview *record annotator*
wview *record annotator start-time*
wview [*drive:*][*path*]*record***.hea**

DESCRIPTION

This program reads one or more signal files and an annotation file (if one is available), and displays the signals with annotations on a PC running MS Windows 3.1 (or later). To run it, either select the DB Browser icon from the appropriate Program Manager group (usually 'Applications'), or select a DB 'header' file (a file named with the suffix '.hea') from the File Manager, or enter one of the *wview* command lines listed above (under 'Synopsis') in the **Run...** window of the Program Manager's **File** menu. The first two of these methods work only if *wview* has been installed as described below, under 'Installation'; use the **Run...** method if you have not performed the full installation procedure. If MS Windows is not running, you may start it and launch the DB Browser from the MS-DOS prompt by typing **win wview** (this command may also include *record*, *annotator* and *start-time* options as shown above). Refer to Microsoft's *Windows User's Guide* for general information about starting and using MS Windows.

The current version of the DB Browser is an alpha release. A few of the controls are not yet implemented, and appear as inactive (greyed) menu items. In comparison with other available software for viewing annotated DB records, the DB Browser's capabilities lie somewhere between those of *view*(1) (for MS-DOS) and *wave*(1) (for the X Window System). The DB Browser can do anything that *view* can do, and much more, but *view* is likely to be substantially faster than the DB Browser on older PCs that do not have 32-bit CPUs with numeric coprocessors and MS Windows accelerated graphics cards. The DB Browser has most of the display capabilities of *wave* with the exception of 'scope' windows, but it lacks *wave*'s capabilities for annotation editing (these may be added in a future version) and control of external analysis programs, as well as a variety of less important features.

Choosing Files to View

The **Choose Inputs** dialog usually appears when you start the DB Browser. It allows you to select a database *record* to view. If you start the DB Browser using a command line, or by selecting a DB 'header' file from the File Manager, the DB Browser opens the record immediately without showing the **Choose Inputs** dialog. If you supply a record name but omit the annotator name, the DB Browser attempts to find the reference (**atr**) annotations. At any time, you may pop up the **Choose Inputs** dialog by selecting **Open...** from the **File** menu, in order to change the record or annotator names, or the *DB environment* (the *DB path* and the *DB calibration file*).

To select sets of signals and annotations to view, fill in the **Record name:** field (up to 8 characters), and the **Annotator name:** field (up to 3 characters). Leave the *Annotator name:* field empty if the record is unannotated, or if you do not wish to view annotations. Although the DB environment may be changed within the **Choose Inputs** dialog, note that the effects of any such changes are limited to the current session of the DB Browser (other DB applications and future DB Browser sessions are unaffected).

The *DB path* is a list of directories that are searched when you specify a record or annotator name. The DB path is usually initialized by setting the MS-DOS environment variable **DB** before starting MS Windows, using the MS-DOS batch file **dossetdb.bat** (see *setdb*(1)). If a DB 'header' file was selected (either from the File Manager or from the command line), the optional *drive:path* component of the file name, if present, is inserted at the beginning of the DB path (but after the current directory, if the DB path begins with the current directory or has not been initialized).

The *DB calibration file* is a text file containing information about the relative scales of many different types of signals (see *dbcal*(5)). The DB calibration file is usually specified by setting the MS-DOS environment variable **DBCAL**; as for the DB path, this is usually performed using **dossetdb.bat** before starting MS Windows. Do not include a drive specification or path information in the name of the DB calibration file unless the file cannot be found in any of the directories named in the DB path.

Browsing a Record

Use the scrollbar at the bottom of the window to move in either direction through the record. If you click on the scrollbar arrows, the display scrolls by 1 second at a time in the indicated direction. Clicking in the rectangles beside the ‘‘thumb’’ of the scrollbar scrolls the display by the width of the window. You may also drag the ‘‘thumb’’ to move to any desired location.

Keyboard commands can also perform these functions and others:

<left-arrow>	back 1 second
<right-arrow>	forward 1 second
<Page Up>	back 1 screenful
<Page Down>	forward 1 screenful
<Home>	back to beginning
<End>	forward to end
<Enter>	search forward (see below)
<backspace>	search backward (see below)
<up-arrow>	zoom in (increase time scale)
<down-arrow>	zoom out (decrease time scale)
+	increase signal amplitude
-	decrease signal amplitude

You may also enter a desired time in the **Recenter at** field of the **Find** window (accessible from the **Edit** menu). The DB Browser redraws the display centered on the specified time.

The times shown in the lower corners of the signal window usually indicate elapsed time from the beginning of the record in hours, minutes, and seconds (hours are omitted if the elapsed time is less than one hour). If the times are shown enclosed in square brackets, they indicate the actual time of day (and possibly the date) when the signals shown were recorded, determined by reference to the base time and date recorded in the header file for the record.

Options

Use the **Options** menu to set display and printing options. The **View Options...** dialog allows you to choose display scales, turn the grid on and off, and select options for annotation and signal display. The **Print Options...** dialog is very similar, but its options apply to printed output only. (The **Print Options...** dialog may also be accessed via the **Options...** button in the **Print...** dialog, which may be opened from the **File** menu.)

Select **Use default scales and options** from **View Options...** to reset all display options to their initial values. In **Print Options...**, select **Use display scales and options** to set the printing options to match those you have chosen for display.

If displayed or printed, the **grid** marks 200 ms intervals horizontally, and 0.5 mV intervals vertically. (For signals not dimensioned in units of mV, the DB calibration file specifies the scaling factor between the physical units of the signal and mV.) At the default scales, the grid intervals should measure 5 mm on your screen or printed output, corresponding to the standard ECG display scales of 25 mm/s and 10 mm/mV. (If this is not the case, click on **Calibrate...** and follow the instructions on-screen to make appropriate adjustments for your hardware.)

Marker bars, if displayed or printed, show the exact locations of each annotation. Note that policies for placement of ECG annotations may vary between records (for example, in the MIT-BIH Arrhythmia Database, QRS annotations are placed at the R-wave peak, but in the AHA Database, these annotations are placed at the PQ junction).

Normally, the DB Browser shows only the mnemonic corresponding to the primary annotation type (the **anntyp** field) of each annotation. Exceptions include **RHYTHM** annotations (for which the rhythm, encoded in the **aux** field of the annotation, is shown); **NOISE** annotations, for which signal quality data encoded in the **subtyp** field are shown); and **STCH**, **TCH**, and **NOTE** annotations (for which the contents of the **aux** field are shown). To make it easier to identify these exceptions, **RHYTHM** annotations appear below the level of ordinary annotations, and the others appear above the level of ordinary annotations. The representation matches that used in the *MIT-BIH Arrhythmia Database Directory* and other directories, which contain complete lists of the mnemonics used in each database.

By selecting the appropriate items from **View Options...** or **Print Options...**, you can view or print the contents of the optional annotation fields (**subtyp**, **chan**, **num**, and **aux**, shown from top to bottom in that order if two or more are selected). See the *ECG Database Programmer's Guide* for further information about optional annotation fields.

Signal baselines, if selected, are displayed or printed only for signals for which absolute levels are significant, such as blood pressure. Such signals are referred to as DC-coupled signals, since they must be digitized without being passed through high-pass filters in order to preserve absolute levels. In signals such as ECGs, only variations in level, rather than absolute levels are significant. These AC-coupled signals are high-pass filtered before digitization, in order to remove any DC component, so that the gain can be chosen optimally for the range of variation in the signal. By default, **signal names** appear at the left margin slightly above each signal; deselect **Signal names** to suppress this output. The header file for each record specifies signal names, and which signals are DC-coupled; see the *ECG Database Programmer's Guide* for further information.

Searching for Annotations

Select **Find...** from the **Edit** menu to bring up the **Find** dialog. Enter an annotation mnemonic (e.g., **V**) in the **Search forward** or **Search backward** field. Click on the radio button to the left of the chosen search direction, then click on **OK**. The DB Browser redraws the signals, centered on the next annotation of the specified type that was not visible in the previous screenful. If the search is unsuccessful, the **Find** window remains visible, so that you can modify the search criteria.

In addition to standard annotation mnemonics (listed in the *ECG Database Programmer's Guide*), you may enter signal quality strings (as displayed by the DB Browser for **NOISE** annotations), or **aux** strings (as displayed for **RHYTHM**, ST and T change, and **NOTE** annotations). In short, you may search for any string that the DB Browser uses when drawing annotations. When specifying **aux** strings, a prefix is sufficient as a search target (for example, "**(S**" matches any annotation with an **aux** string beginning with these characters, such as "**(SVTA**" or "**(ST0+**").

To repeat a search forward, press the <enter> key; to repeat a search backward, press the <backspace> key. If you use these keyboard shortcuts without having defined a search target, <enter> behaves in the same way as <Page Down>, and <backspace> in the same way as <Page Up> (moving to the adjacent screenful in the appropriate direction).

Printing

You can print the current contents of the main DB Browser window, or any selected segment of the current record, by selecting **Print...** from the **File** menu to bring up the **Print** dialog. Choose the output device from the **Printer** list. Select the range (time interval) to be printed by choosing either **Entire Record**, **Current Contents of Window** (the default), or **Segment**. If you choose **Segment**, enter the times of the beginning and end of the desired segment in the **From:** and **To:** fields. Press **OK** to begin printing, or **Cancel** to return to the DB Browser without printing.

Help

On-line help is available by selecting a topic from the **Help** menu, or by selecting any control and pressing the **F1** key. The **Help** menu topics include most of the text of this *man* page.

ENVIRONMENT

DB 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 *db*(3) library search for their database input files in the order specified by **DB**. If **DB** is not set, searches are limited to the current directory. Under MS-DOS, directory names are separated by semicolons (;), and the format of **DB** is that of the MS-DOS **PATH** variable (colons may be used following drive specifiers within **DB** in this case).

DBCAL

The name of the DB calibration file (see *dbcal*(5)), which must be in a directory named by **DB** (see immediately above). This file is used by the DB Browser to determine standard scales for signals other than ECGs. If **DBCAL** is not set, or if the file named by **DBCAL** is not readable, these signals may be drawn at incorrect scales.

After determining appropriate values for these variables, you may wish to add commands for setting them to your *autoexec.bat* file. The standard installation procedure determines appropriate values interactively and inserts the necessary commands into *dossetdb.bat* (see *setdb*(1)).

INSTALLATION

The current version of the DB Browser requires manual installation of a few files:

wview.exe	the DB Browser itself, in Windows executable form
wview.hlp	the compressed MS Windows Help file for the DB Browser
db.dll	the DB library, compiled as a large model MS Windows DLL

All of these files should be installed in the same directory, which should be somewhere in your **PATH**; the main MS Windows directory (usually **c:\windows**) seems to be the most popular choice for commercial MS Windows applications, and is suitable for this purpose. Be certain that you do not overwrite other files of the same names, however (I don't know of any commercial applications that use these file names, but check your system to be safe). At this point, it is possible to start the DB Browser using the command-line interface.

To install the DB Browser icon in the Program Manager workspace, select (click once on) a program group (such as 'Applications'). Select **New...** from the Program Manager's **File** menu, choose **Program item** from the **New Program Object** dialog that appears, and click on **OK**. The **Program Item Properties** dialog then appears. Enter "**DB Browser**" in the **Description:** field, and the full pathname of **wview.exe** in the **Command line:** field, then click on **OK**. It is now possible to start the DB Browser by clicking on its icon.

Finally, open the MS Windows File Manager and choose **Associate...** from its **File** menu. Enter "**hea**" in the **Files with extension:** field, and click on **Browse**. Find and select **wview.exe** using the **Browse** dialog, and click on **OK**. This procedure makes it possible to start the DB Browser simply by double-clicking on any DB header (**.hea**) file.

BUGS

Probably many. Please send your comments, suggestions, and bug reports to the author:

George B. Moody
MIT Room 20A-113
Cambridge, MA 02139 USA
Internet: george@hstbme.mit.edu

A significant known bug is that only one instance of the DB Browser may be run at any given time. This is a consequence of using a large memory model DLL (in this case, **db.dll**) under MS Windows 3.1. Fixing this bug is a very low priority (i.e., not likely to happen soon unless it is fixed by a future version of MS Windows). If you are careful, it is possible to have two or more instances running simultaneously *provided that the same record (and annotator, if any) are open in all instances*. Exit cleanly by iconifying (closing) all instances first, then quit each instance without reopening the window.

SEE ALSO

dbplot(1) (for UNIX), dbtool(1) (for SunView), pschart(1) (for PostScript), view(1) (for MS-DOS), wave(1) (for X11)

AVAILABILITY

The DB Browser (*wview*) is not included in the DB Software Package, but is available separately. Please do not redistribute copies of alpha versions of the DB Browser. Refer anyone interested in obtaining a free copy to the author at the address above.

NAME

xform – sampling frequency, amplitude, and format conversion for DB 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.

-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 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 *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, separated by spaces; default: write all signals). This option may be used to re-order or duplicate signals.

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

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, provided that both are integers (if non-integer sampling frequencies are specified, they are truncated to multiples of 1 Hz before the interpolation/decimation parameters are determined). 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*.

The shell variable **DB** should be set and exported (see *setdb*(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), setdb(1), snip(1), signal(5)

NAME

db − ECG database library

SYNOPSIS

#include <ecg/db.h>

int adumuv(DB_Signal *s*, DB_Sample *adc_units*)
double aduphys(DB_Signal *s*, DB_Sample *adc_units*)
char *anndesc(int *annotation_code*)
int annopen(char **record*, DB_Anninfo **aiarray*, unsigned int *nann*)
char *annstr(int *annotation_code*)
int calopen(char **calibration_filename*)
char *datstr(DB_Date *d*)
char *dberror(void)
char *dbfile(char **type*, char **record*)
void dbflush(void)
int dbgetskew(DB_Signal *s*)
long dbgetstart(DB_Signal *s*)
int dbinit(char **record*, DB_Anninfo **aiarray*, unsigned int *nann*,
DB_Siginfo **siarray*, unsigned int *nsig*)
void dbquiet(void)
void dbquit(void)
void dbsetskew(DB_Signal *s*, int *skew*)
void dbsetstart(DB_Signal *s*, long *byte_offset*)
void dbverbose(void)
char *ecgstr(int *annotation_code*)
void flushcal(void)
int getann(DB_Annotator *a*, DB_Annotation **annotation*)
double getbasecount(void)
int getcal(char **description*, char **units*, DB_Calinfo **cal*)
DB_Frequency getcfreq(void)
char *getdb(void)
int getframe(DB_Sample **vector*)
char *getinfo(char **record*)
int getspf(void)
int getvec(DB_Sample **vector*)
void iannclose(DB_Annotator a)
int iannsettime(DB_Time *t*)
int isgsettime(DB_Group *signal_group*, DB_Time *t*)
int isigopen(char **record*, DB_Siginfo **siarray*, int *nsig*)
int isigsettime(DB_Time *t*)
char *mstimstr(DB_Time *t*)
DB_Sample muvadu(DB_Signal *s*, int *microvolts*)
int newcal(char **calibration_filename*)
int newheader(char **record*)
void oannclose(DB_Annotator a)
int osigfopen(DB_Siginfo **siarray*, unsigned int *nsig*)
int osigopen(char **record*, DB_Siginfo **siarray*, unsigned int *nsig*)
DB_Sample physadu(DB_Signal *s*, double *v*)
int putann(DB_Annotator *a*, DB_Annotation **annotation*)
int putcal(DB_Calinfo **cal*)
int putinfo(char **info*)
int putvec(DB_Sample **vector*)
DB_Frequency sampfreq(char **record*)
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(DB_Frequency *counter_frequency*)
void setdb(char **database_path_string*)
int setecgstr(int *annotation_code*, char **annotation_mnemonic_string*)
void setgvmode(int *mode*)
int setheader(char **record*, DB_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(DB_Frequency *sampling_frequency*)
int strann(char **annotation_mnemonic_string*)
DB_Date strdat(char **date_string*)
int strecg(char **annotation_mnemonic_string*)
DB_Time strtim(char **time_string*)
char *timstr(DB_Time *t*)
int ungetann(DB_Annotator *a*, DB_Annotation **annotation*)

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 DB library. Under UNIX, programs may be linked with the DB library by using the **-ldb** 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/local/lib/libdb.a standard (statically bound) library

/usr/local/lib/libdb.so

/usr/local/lib/libdb.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/local/lib/libdb.sa stubs for linking with applications that use **libdb.so** (not needed on all systems).

The location of these files may vary on some systems.

MS-DOS/MS Windows systems:

db.lib standard (small memory model) library

dbl.lib large memory model library

db.dll dynamic link library for MS Windows

dbdll.lib

stubs for linking with applications that use **db.dll**

SEE ALSO

ECG Database 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/local/help/html/dbpg*); the most recent version can be viewed at **http://ecg.mit.edu/dbpg/** if you have Internet access.

The DB library can also be used with Fortran programs; see 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.

NAME

annot − ECG database annotation file formats

SYNOPSIS

#include <ecg/ecgcodes.h>

DESCRIPTION

Programs compiled with the DB library (**-ldb**) 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. DB 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 **<ecg/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 **<ecg/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 DB library applications (such as those found on MIT-BIH database distribution tapes), 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

ECG Database Programmer's Guide

NAME

dbcal − ECG database calibration file format

DESCRIPTION

Programs compiled using the DB library (see *db*(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. *calibrate*(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 DB 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 DB library function *getcal*, used by programs such as *calibrate*(1), *dbplot*(1), and *psfd*(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 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'.

ENVIRONMENT

Programs compiled with the DB library use the environment variable **DBCAL** to determine the name of the calibration file. Calibration files must be located in one of the directories named by the environment variable **DB** (see *setdb*(1)).

BUGS

Support for calibration files in this format was added to the DB library in version 6.0. Programs compiled with earlier versions of the DB library must be modified and recompiled in order to use the features described above.

SEE ALSO

calibrate(1), setdb(1), header(5)
ECG Database Programmer's Guide

NAME

header − ECG database 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 DB library (*-ldb*) can read header files created by *newheader* (see *db*(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 DB 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 DB library version 9.1 and later versions) contain a *segment specification line* for each segment.

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 DB 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 *<ecg/db.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. DB 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. DB 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 DB 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 **DB** (the database path) lists the directories in which signal files (as well as DB header and annotation files) are found; normally **DB** 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 **DB**, that directory is appended to the end of **DB** (by functions that read header files in DB 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 *<ecg/db.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.) DB 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). DB 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 DB library; the DB library does not support any means of writing such files, and byte offsets must be inserted into header files manually. DB 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 *<ecg/db.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. DB 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. DB 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 DB library functions that read header files supply a description of the form "record *xxx*, signal *n*".

Segment specification lines

Each non-empty, non-comment line following the record line in a multi-segment record contains specifications for one segment, beginning with segment 0. 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 DB 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 DB 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), and the number of samples per signal must be defined for each segment in the record line of the segment's own header file. In addition, the number of signals and the sampling frequency should match in all segments of a record, and it is best if 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 DB 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 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.

Info strings

Comment lines that follow the last signal specification line in a header file can be read and written by the DB 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*.

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, DB library-based programs will find the signal file only if it is located in one of the directories specified by the **DB** 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 **DB** 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 DB 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 DB application built using DB library version 9.1 or later; the application need not be aware that this is a multi-segment record. Earlier versions of the DB library do not support multi-segment records (or format 0 signals).

Old format

Versions 2.3 through 4.6 of the DB library included support for reading header files written in an obsolete format. This support has been removed from DB library version 5.0. Obsolete-format header files can be brought up-to-date using *revise* (in the *convert* directory of the DB software distribution).

SEE ALSO

ECG Database Programmer's Guide

NAME

signal − ECG database signal file formats

DESCRIPTION

ECG database 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 the samples read from a format 8 file may not always be identical to those read from an equivalent format 16 file. On-line signal files on magnetic disk are often kept in format 8 to save space. 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. The format used for MIT−BIH and AHA database distribution 9-track tapes is format 16, with the addition of a logical EOF (octal 0100000) and null-padding after the logical EOF.

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 on the second edition of the MIT−BIH Arrhythmia Database CD-ROM are format 212 files.

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). The entire process is then repeated for each successive set of three samples.

SEE ALSO

header(5)
ECG Database Programmer's Guide

Installing the DB Software Package

Obtaining the DB Software Package

Several versions of the DB Software Package, differing in their completeness, are available. From our web site (`http://ecg.mit.edu`) you may obtain up-to-date binary versions of the DB Software Package for various popular operating systems. (Currently these include x86 Linux, Sparc Solaris, Sparc SunOS, and MS-DOS/MS-Windows.) The web site also offers selected portable C sources, including those for the DB library, for the `rdann`, `rdsamp`, and `xform` utilities, and for all of the ANSI/AAMI EC38 required evaluation software. If you have been using earlier versions of this software included on the first or second editions of the *MIT-BIH Arrhythmia Database CD-ROM*, the *MIT-BIH Polysomnographic Database CD-ROM*, the *European Society of Cardiology ST-T Database CD-ROM*, or the *Massachusetts General Hospital/Marquette Foundation Waveform Database CD-ROMs*, you should update your software by downloading the newer versions available from our web site (or from one of our more recent CD-ROMs, see below). The web site also offers demo versions of WAVE for Linux, Solaris, and SunOS (saving the results of annotation editing operations is not possible using these demonstration versions). If you prefer, you may obtain the same materials by anonymous FTP from `ecg.mit.edu`; download and read `pub/software/README.TXT` first to determine what files you will need; follow the instructions in that file for creating a `software` directory on your hard disk.

The complete DB Software Package (both sources and binaries for Linux, Solaris, SunOS, and MS-DOS/MS-Windows) is included on the *MIT-BIH Arrhythmia Database CD-ROM* (third edition), and also on the *Software for Physiologic Databases with Samples CD-ROM*, available from MIT (see `http://ecg.mit.edu/` for information about these CD-ROMs). These CD-ROMs also include fully functional versions of WAVE for Linux, Solaris, and SunOS (binaries only). Installation instructions are included in `software/README.TXT` on these CD-ROMs.

Choosing an installation method

From within the `software` directory of your hard disk (if you have downloaded the DB Software Package from our web site or FTP server), or from within the `software` directory of your CD-ROM, follow the instructions appropriate for your operating system.

If you use Linux, SunOS, or Solaris

Examine the script `install.unx` in the `software` directory, then run it (you will need root permissions to do so successfully on most systems). This script will identify your operating system and install the appropriate binaries in `/usr/local/bin` (other files needed by the DB Software Package will be installed in other subdirectories of `/usr/local`).

If you have been using a fully functional version of WAVE from one of our CD-ROMs, be careful not to overwrite it with a demo version of WAVE if you have downloaded the software from our web site or FTP server. The simplest way to avoid this possibility is to rename your old version of `wave` as `wave-old` before running `install.unx`.

If you wish to customize or recompile the DB Software Package, follow the instructions in the next paragraph, for users of other Unices.

If you use another version of Unix

The DB Software Package has been successfully compiled on dozens of Unix variants, using both ANSI/ISO and K&R C compilers.

Copy the source files (in the `software/db` directory) to a working directory on your hard disk, then examine `db/Makefile` for information about compiling the DB Software Package on your system. In most cases, you will need to change a few settings in `db/Makefile` to configure the software for your system, then type `make` to compile the software, and `make install` to install it. You will need root permissions to install the software in the standard locations.

WAVE is not part of the DB Software Package, but is intended to work with this package on systems that support X11 and XView (currently Linux, Solaris, and SunOS). If you would like to use WAVE on another

UNIX system, you will need to port XView to your system first (or purchase a commercial port if one is available). Sources for XView are supplied on our CD-ROMs, and are also available from sunsite.unc.edu, tsx-11.mit.edu, 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).

If you use MS-DOS or MS-Windows

Run install.exe from the software directory. Follow the on-screen directions to choose the installation directories, to set up your DB path, and to calibrate your display. Detailed instructions may be found in software/MSDOS.TXT.

The source files within the db directory are in UNIX text format (newlines are marked by ASCII line feed characters only). Most MS-DOS C and C++ compilers and text editors can process UNIX-format text files without difficulty; if you wish to convert these files to MS-DOS native text file format, however, first install msdos/bin/u2d.exe into a directory in your PATH, copy the UNIX-format files to a writable directory, change to that directory, and type 'u2d *.*'. (You may name individual files to be converted if you prefer. 'u2d' does not modify binary files or files that are already in MS-DOS text format.)

If you wish, you can compile the sources using Microsoft or Borland C or C++ compilers without modification; see the files named makefile.dos in the subdirectories of 'db' for details. If you have Microsoft or Turbo C or C++, and a Microstar Laboratories DAP 1200- or 2400-series analog interface board, you can recompile sample (a program for creating database records from analog signals, and for replaying them in analog form). To do so successfully, you must first have installed the appropriate Microstar #include files and DAP interface library for use with your C compiler. Specifically, files c_lib.c, clock.h, and ioutil.h must be installed in your include directory, and file cdapl.lib must be installed in a directory in which libraries are found by your linker.

If you use a Macintosh or another system

The DB Software Package is written in highly portable C, and (with the exception of a few MS-DOS or UNIX-specific display or data-acquisition programs) should be easy to compile with any K&R or ANSI C compiler. The UNIX and MS-DOS make description files (Makefile and makefile.dos in software/db and in each of its subdirectories) should get you started.

Notes for Macintosh users can be found in software/MAC.TXT, and in software/db/MAC.TXT. These include detailed instructions for compiling the DB library using Symantec's Think C. Since most of the applications are command-line oriented, they will require minor modifications to run under the Mac OS.

Your comments are requested

I would greatly appreciate a report of any problems you encounter in installing or using this software, if possible by e-mail to george@hstbme.mit.edu, or to:

George B. Moody
MIT Room 20A-113
Cambridge, MA 02139 USA

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 EC-38, 1994); 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 DB Software Package available from MIT; this suite of software includes the reference implementations of the evaluation protocols specified in EC-38.

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.

At present, the following ECG databases are available:[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; supplied with the MIT DB)

- **CU DB**: The Creighton University Sustained Ventricular Arrhythmia Database (35 records, 8 minutes each; supplied on the second edition of the MIT DB CD-ROM)

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 1994, the ambulatory ECG subcommittee of the AAMI ECG committee developed 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 (EC-38) builds 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 EC-38 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, EC-38 specifically requires that they be pro-

[1] Sources: ECRI, 5200 Butler Pike, Plymouth Meeting, PA 19462 USA (AHA DB); MIT-BIH Database Distribution, MIT Room 20A-113, 77 Massachusetts Avenue, Cambridge, MA 02139 USA (MIT, ESC, NST, and CU databases). Outside of North America, the ESC DB is available from: CNR Institute of Clinical Physiology, Computer Laboratory, via Trieste, 41, 56100 Pisa, Italy. Except for the AHA DB, all are available in CD-ROM format.

[2] *Testing and Reporting Performance Results of Ventricular Arrhythmia Detection Algorithms.* Publication AAMI ECAR (1987); available from AAMI, 3330 Washington Boulevard, Suite 400, Arlington, VA 22201 USA.

[3] *American National Standard for Ambulatory Electrocardiographs.* Publication ANSI/AAMI EC-38 (1994); available from AAMI (address above).

duced 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 EC-38 standard (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 DB Software Package from MIT[4] and are included in the current edition of the MIT-BIH Arrhythmia Database CD-ROM. In this paper, the names of these programs are printed `like this`.

[4]Source: MIT-BIH Database Distribution (address above).

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.

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 DB software package includes a program (`sample`) that uses a Microstar DAP 2400-series analog interface board[5] 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.[6] The DB library (included in the DB 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.[7]

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 `<ecg/ecgcodes.h>`. (This file is supplied with all of the existing CD-ROM databases.) 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 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 ventric-

[5]Source: Microstar Laboratories, 2863 152nd Avenue NE, Redmond, WA 98052 USA. 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.

[6]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).

[7]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.

ular 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, "`(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.

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.[8]

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.[9] 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

[8]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.

[9]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).

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 DB 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![10]

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 DB 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.

- `epic` 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.[11]

The DB Software Package also includes three related programs:

- `sumstats` reads certain output files generated by `bxb`, `rxr`, `epic`, and `mxm`, and calculates aggregate statistics for a set of records.

- `plotstm` generates scatter plots of ST deviation measurements collected by `epic`.

- `ecgeval` automates the entire comparison procedure by running `bxb`, `rxr`, `epic`, 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 *ECG Database Applications Guide*, which accompanies the DB 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 ECAR and EC38 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

[10]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.

[11]`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`.

(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")[12] 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
epic -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.[13] `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 `epic` 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
```

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 `view`, `dbplot`, `dbtool`, `wave`, and `wview`[14] 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 DB 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 pro-

[12]Annotation files for any given record are distinguished by annotator names, which, under MS-DOS, correspond to the "extension" of the file name. The reference annotation files supplied with the databases have the annotator name "`atr`" (from "`atruth`", as used in environments with less restrictive file-naming schemes, originally because "`a`" was intended to indicate the file type, and "`truth`" because ... well, because the annotations are supposed to be The Truth).

[13]`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.

[14]`view` (for MS-DOS), `dbplot` (for a variety of graphics devices under UNIX), and `dbtool` (for SunView) are included in the DB Software Package. The DB Software Package, and binary versions of `wave` (for SunOS, Solaris, and Linux) and of `wview` (for MS Windows), are included on the third edition of the MIT-BIH Arrhythmia Database CD-ROM.

tocols, 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 CD-ROM format, the DB Software Package includes a pair of programs that convert files read from AHA DB distribution tapes or floppy disks into files formatted in the same way as those in the CD-ROM databases. `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 DB library; refer to the *ECG Database 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 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 DB 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).