NAME

frmfile − frame file format

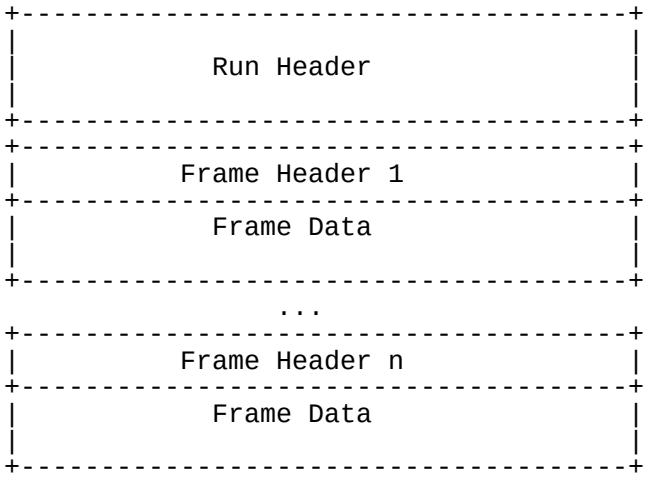
SYNOPSIS

#include "/usr/neuro/src/run.h"

DESCRIPTION

The frame file is the key file in the set of data, collectively known as a *runfile*, produced or used by *dsepr*(1), *analysis*(1) and related programs. It is the one that holds information on both triggered and untriggered channels, as well as the data collected from the triggered channels.

Frame File Format



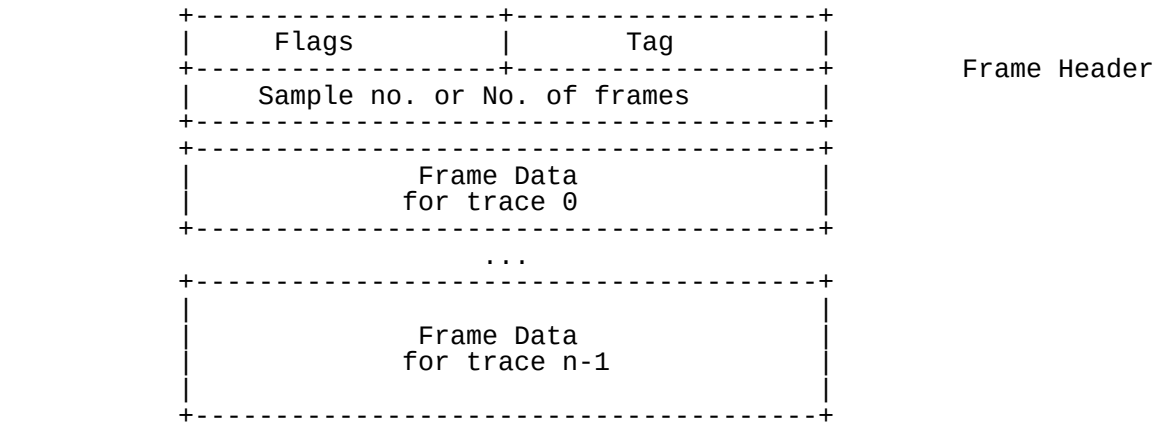
Files from a Captured Run

Each run of data results in one frame file, and a waveform file for each untriggered channel. The frame file will always begin with a run header, which records general information about the run, such as sampling rate, number of frames, and for each channel, the sampling rate divisor, and calibration information. Captured frames of triggered channel data follow. Each frame begins with a frame header which indicates the starting sample number (i.e. at what time within the run the trigger pulse occurred). The frame also contains one trace (a series of A/D samples) for each triggered channel. Each waveform file contains a contiguous series of A/D samples for one untriggered channel.

Frame File for Averaged Data

Averaged data, produced by either real-time or post-hoc averaging, are stored in a frame file of the same format as that used for raw data. A value in the run header indicates the method used to obtain the average. One frame is recorded for each bin in a multi-bin average. Instead of recording the starting sample number, the frame header records the number of sweeps of raw data that were averaged in that bin.

Frame Format



Frame Header

The frame header has three fields: deletion flags, a tag code, and a number. This number is either a sample number indicating when the raw frame was triggered, or a count of the number of sweeps averaged into the frame.

Frame Data

The sample points for one triggered sweep from each triggered channel are stored after the frame header. Each sweep is stored contiguously. Since channels can have independent sampling rate divisors, the sweeps in a frame may not all contain the same number of data points, even though they all represent the same length of time. However, the number of points per sweep for any given triggered channel is consistent throughout all frames.

Binary Data Formats

A/D data samples in the frame file and waveform files are stored as 16 bit signed integers, 2's complement for negatives, and most significant byte first (known as big-endian). Numbers in the run header and frame headers are either long integers (32-bit, 2's complement signed, big-endian), short integers (16-bit, 2's complement signed, big-endian), double precision floating-point (64-bit, IEEE format, big-endian), or character (fixed length arrays, single-byte ASCII, NUL terminated).

Below are the C Language structure definitions used for the headers. They are appropriate for big-endian architectures (Motorola, SPARC, PowerPC). For little-endian architectures (DEC, Intel) it is necessary to flip the byte ordering for all short ints, long ints and doubles.

Listing 1: run.h

```
/*
 * run.h -- definitions of run file headers and related info.
 *
 * (c) 1988, G.R. Detillieux, Spinal Cord Research Centre,
 * The University of Manitoba.  All Rights Reserved.
 */

#define NCHFRM          16              /* # chans for frame (triggered) data */
#define NCHREG          16              /* # chans for regular (untriggered) data */

#define CHNAMESIZ       42              /* size of channel name string */

typedef short int       ADSAMPLE;       /* data type used for A/D values */
typedef long int        ADPERIOD;       /* data type used for time values (sampling periods) */
```

```
/* calibration info for one channel: */
struct calinfo {
        ADSAMPLE        ca_zero;        /* A/D value for zero volts */
        ADSAMPLE        ca_height;      /* A/D value for cal pulse minus value for zero */
        long int        ca_level;       /* level of cal pulse in uV */
        short int       ca_gain;        /* gain factor used for channel */
        char            ca_name[CHNAMESIZ];     /* channel name string */
};


#define RUNMAGIC        0xFFAAFABF      /* "magic" number identifying run file */

/* define averaging methods used by dsepr, others in analysis/param.h: */
#define AM_TRAWCAP      0       /* raw trace capture */
#define AM_TAVGCAP      1       /* averaged trace capture, ...    */
                                /* ... same code as AM_TAVGFRMLST */

/* run header in frame data file: */
struct runhdr {
        long int        rh_magic;       /* magic number for run file */
        ADPERIOD        rh_length;      /* run length as #samples */
        double          rh_samprate;    /* sample rate in Hz */
        long int        rh_nframes;     /* # frames in file */
        long int        rh_frmsiz;      /* frame size in bytes */
        ADPERIOD        rh_delay;       /* delay to start of frame as #samples */
        ADPERIOD        rh_window;      /* frame sampling window as #samples */
        ADPERIOD        rh_gpper;       /* gate pulse period as #samples */
        ADSAMPLE        rh_minbinlevel; /* min. W.F. level for AM_TAVGLEVEL */
        ADSAMPLE        rh_maxbinlevel; /* max. W.F. level for AM_TAVGLEVEL */
        short int       rh_avgmethod;   /* averaging method, or 0 for raw */
        short int       rh_levelwf;     /* W.F. from which above levels come */
        ADPERIOD        rh_wreduce;     /* #samples by which usable window reduced */
        long int        rh_starttime[2];/* time capture started, if available */
        short int       rh_reserve[19]; /* reserved for future use */
        short int       rh_needrhdfile;         /* extra run header info. needed */
        short int       rh_npts[NCHFRM];        /* # points in triggered channel frames */
        short int       rh_frmdiv[NCHFRM];      /* sample rate divisors for trig. chan. */
        short int       rh_regdiv[NCHREG];      /* sample rate divisors for untrig. chan. */
        short int       rh_frmchan[NCHFRM];     /* channel numbers for traces */
        short int       rh_regchan[NCHREG];     /* channel numbers for waveforms */
        struct calinfo  rh_frmcal[NCHFRM];      /* calibr. info for framed channels */
        struct calinfo  rh_regcal[NCHREG];      /* calibr. info for regular (unframed) channels */
        long int        rh_frmres[NCHFRM];      /* reserved for future use */
        long int        rh_regres[NCHREG];      /* reserved for future use */
};


/* header at beginning of each frame: */
struct frmhdr {
        long int        fh_flags;       /* tag, deletion flags & room for more */
        ADPERIOD        fh_sampnum;     /* sample # where gate pulse detected */
};

#define fh_numsweeps    fh_sampnum      /* # of sweeps in an averaged frame */

/* frame header flags: */
#define DELFLAGS        0xE0000000      /* all 3 deletion flags */
#define MANDEL          0x80000000      /* frame manually deleted */
#define AUTODEL1        0x40000000      /* frame auto-del'd due to clipping */
#define AUTODEL2        0x20000000      /* frame auto-del'd due to bad cal pulse */
#define ISDELETED(fh)   ((fh).fh_flags & DELFLAGS)
#define TAGMASK         MAXSHORT        /* all bits reserved for tags */
#define NTAGS           4096            /* # of different tags used */
#define INTAGLIST(fh,tlist,tlen)        (inlist((unsigned short int)((fh).fh_flags & TAGMASK), \
                                                (tlist), (tlen)))
```

Calibration and unit conversions

Note that the fields rh_frmcal and rh_regcal are both arrays of structures containing the calibration information and identifying name for each channel. The rh_frmcal is used for the framed, or triggered channels, whose samples are contained in the frames of this same file, right after the run header. The rh_regcal is used for the regular, or untriggered channels, whose samples are contained in the separate waveform files.

These structures are copied from the calibration file at the start of the data capture. The calibration file (default.cal) simply contains an array of 16 or more of these structures, one for each channel on the A/D

converter, with no header or other fields in the file. The structures for the channels to be captured are copied to the rh_frmcal and rh_regcal entries corresponding to the traces or waveforms to which these channels are assigned.

Each of these structures contains three fields that permit conversion from A/D sample values to voltages. The ca_zero represents a zero volt level on the channel, expressed as an A/D sample value. The ca_height represents the height of a calibration pulse of known amplitude, expressed as a displacement in A/D sample values. The ca_level is the amplitude of this same calibration pulse in microvolts. Thus, to convert A/D samples for a given untriggered channel "i" to millivolts, one could use this formula:

$$mv = \frac{(samp - (ca\_zero \text{ of } rh\_regcal_i)) \times (ca\_level \text{ of } rh\_regcal_i)}{(ca\_height \text{ of } rh\_regcal_i) \times 1000}$$

the equivalent in C Language, with required type conversions, would be:

```
mv = (samp-run.rh_regcal[i].ca_zero)
                * (double)run.rh_regcal[i].ca_level
        / ((double)run.rh_regcal[i].ca_height * 1000.0);
```

Two other fields in the calibration information structures are ca_gain and ca_name. The ca_gain is simply the gain code used for the A/D channel used for this signal. It is provided for information only, and is not required for level conversions of the A/D samples. The ca_name is an ASCII NUL terminated character string containing the name given to the channel − usually describing the signal.

Similar unit conversions must be done on the time axis, to convert from sample numbers to, say, milliseconds. The rh_samprate contains the base sampling rate at which the run was captured, so to calculate the the time of onset of the n-th sample at this rate, in milliseconds, the C formula would be:

```
ms = n * 1000.0 / run.rh_samprate;              [1]
```

Complicating matters somewhat is the fact that samples are not necessarily stored at the base sampling rate for all channels. In fact, the channels may all be stored using different effective rates. The rh_frmdiv and rh_regdiv arrays contain the sampling rate divisors for all triggered and untriggered channels, respectively. These divisors are all integers, and determine how samples are stored for the channels. A divisor of 1 indicates that all samples are stored, at the base sampling rate. A divisor of n indicates that only the first of every n samples at the base rate is kept for the channel. As a special case, a 0 divisor means this particular channel was not used.

The complete C formula to calculate the time of onset of the n-th stored sample for untriggered channel "i" is:

```
ms = n * 1000.0 / (run.rh_samprate
                    / run.rh_regdiv[i]);        [2]
```

The sampling rate division of triggered channels is handled similarly, except that the whole process is restarted for each frame. Trigger pulses on the trigger signal (which is not usually stored in the run) cause frames to be started, triggering sweeps from each of the triggered channels. When a trigger pulse occurs during the capture of a raw run file, the time of onset of the trigger is recorded in the fh_sampnum field of the new frame's header, as a sample number at the base sampling rate. Thus, the first time conversion formula above [1] can be used to get the time of onset in milliseconds.

From the time of onset of the trigger pulse, the capture program waits for a length of time specified by the rh_delay field, then begins storing samples from the triggered channels into the new frame, for the length of time specified by the rh_window field. Both fields specify time as a number of samples at the base sampling rate. A negative delay indicates that pre-trigger sampling was performed in the frames. For any triggered channel with a divisor n greater than 1, only the first of every n samples is kept, starting with the first sample after the delay has passed. Since triggered channels can be stored at different rates, but for the same length of time in each frame, the number of sample points per frame can vary from channel to channel. The rh_npts array indicates the number of samples per frame for each triggered channel.

The time offset of the n-th sample of channel "i" from the start of the frame can be calculated with a variant of formula [2], using rh_frmdiv rather than rh_regdiv. The time offset of the n-th sample in a given frame, from the start of the run is:

```
ms = (frm.fh_sampnum + run.rh_delay
                + (n * run.rh_frmdiv[i]))
        * 1000.0 / run.rh_samprate;            [3]
```

Extended run header file format

For runs of data that have more than 16 waveforms, more than 16 traces, or calibration zero or height values that are larger than 16 bits, the run header format described above is not sufficient. Rather than changing the format, which would break backwards compatibility, we use this same header format, in the **.frm** (frame) file, for the first 16 traces and waveforms, but also add another **.rhd** run header file, which is an ASCII text file that contains run header information on all traces and waveforms in use for the run, as well as other parameters common for the entire run. The programs that open a *runfile* will check for the presence of this run header file, and if there they will also verify that the information in it is consistent with the run header in the frame file, for all overlapping parameters.

The contents of this run header file consists of lines of parameter settings, much like shell variables, in the form

```
NAME='value'
```

The names are all upper case variants of the names in the *runhdr* structure shown above, e.g.: **LENGTH**, **SAMPRATE**, **NFRAMES**, **FRMSIZ**, **DELAY**, **WINDOW**, **GPPER**, **MINBINLEVEL**, **MAXBINLEVEL**, **AVGMETHOD**, **LEVELWF**, **WREDUCE** and **NEEDRHDFILE**. (This latter field is set to 1 if the run header file is needed.) There are also variable names in the form **FRMDIV_***n* and **REGDIV_***n* (where *n* is the trace or waveform number, from 0 up to a maximum of 99), for all trace or waveform specific run header parameters. These include **NPTS**, **FRMDIV**, **FRMCHAN**, **FRMCALZERO**, **FRMCALHEIGHT**, **FRMCALLEVEL**, **FRMCALGAIN**, **FRMCALNAME**, **REGDIV**, **REGCHAN**, **REGCALZERO**, **REGCALHEIGHT**, **REGCALLEVEL**, **REGCALGAIN** and **REGCALNAME**. Finally **RESERVED_***n* is set for any of the reserved fields which are non-zero, for the sake of preserving backwards compatibility for any future uses of these fields.

Start time field format

In 2015 we added a rh_starttime field to the run header, to track the actual time the data capture started for a given run. Before this, we could only estimate this based on the modification time of the oldest waveform files for a run, as anything else was subject to subsequent modification. We use a standard Unix-style time_t format data type for this, which records seconds since midnight UTC of January 1st, 1970. The problem with this data type is it is in transition from a 32-bit integer to a 64-bit one, because the 32-bit data type will overflow early in the year 2038. Because not all systems currently use the 64-bit type (Mac OS X was one of the first to make the transition), we define the field as an array of two 32-bit long integers, and handle the machine-dependent conversion in software. (See /usr/neuro/src/lib/runio.c for examples of this.) When displaying this time, e.g. in dumprun, we use the C library's localtime() function to convert to something human-readable.

By storing the time in UTC internally, we have a start time that is globally accurate even when data files travel across time zones. This is in contrast to the start date and time stored by Axoscope or pClamp in the ABF file's header, which is stored in local time. This can lead to some confusion if ABF files travel across time zones and are then converted by axon2run, as the resulting run file's start time will not accurately reflect the true local start time of the ABF file's capture. It is best to convert ABF files in the time zone in which they were captured, which can be overridden in software when running axon2run.

Raw direct-to-disk capture format

Programs like *dcap* and *dcavg*, or Concurrent's *dacq*, produce files in a raw direct-to-disk capture format. These files can be used as input to *dsepr*(1). The format of the data in such a file is simply an array of A/D samples, stored as 16 bit signed integers, 2's complement for negatives, and in the machine's own native byte ordering. The samples are stored in the order they are sampled from the A/D

converter: the first sample from the first channel, then the second channel, third channel and so on, followed by the second sample from each of these channels, and on. There are no headers, and no separators or delimiters, just binary data. Information about the number of channels, sampling frequency, and calibration are not stored directly in these files, but must be kept track of separately. This could be used as an intermediate format for a data conversion program, which could then use *dsepr* to handle all the details of run file creation.

FILES

*.frm	frame files, containing run headers
*.rhd	run header file for extended file format
*.w??	corresponding waveform files
*.txt	corresponding run descriptions
*.frd	corresponding frame descriptions

SEE ALSO

dsepr(1), cap(1), analysis(1), dumprun(1), axon2run(1)