dacqUSB Data File Formats

Axona Ltd
Unit 4U St. Albans Enterprise Centre
Long Spring, Porters Wood
St. Albans, Herts, AL3 6EN
United Kingdom

http://www.axona.com

Table of Contents

File Overview

The Axona *dacqUSB* data acquisition system stores recording trial data in a number of separate files. At the end of a session, the user is given the opportunity to enter a distinguishing name for the dataset. Supposing this name is chosen to be “trial1”, then data files will be called things like “trail1.set”, “trial1.pos”, “trial1.eeg”, and so on.

The currently defined file types include:

data.set	setup file containing all hardware setups related to the trial
data.1	spike times and waveforms for tetrode 1, or stereotrodes 1 and 2
data.2	spikes times and waveforms for tetrode 2, or stereotrodes 3 and 4
…	
data.32	spikes times and waveforms for tetrode 32
data.spk	spikes times and waveforms for monotrodes (single electrodes) 1 to 16
data.eeg	continuous 250 Hz EEG signal, primary channel
data.eegX	continuous 250 Hz EEG signal, secondary channel (X = 1..16)
data.egf	high resolution 4800 Hz version of primary EEG channel
data.egfX	high resolution 4800 Hz version of primary EEG channel (X = 1..16)
data.pos	tracker position data
data.inp	digital input and keypress timestamps
data.stm	stimulation pulse timestamps
data.bin	raw data file
data.epp	field potential parameters
data.epw	field potential waveforms
data.log	DACQBASIC script optional user-defined output files

Common format elements

Other than “.set” setup and “.log” files which are entirely ASCII text, the data files consist of a readable ASCII text header followed by data in binary form. The ASCII text will vary according the data file type, but will always end in this string of characters:

```
data_start
```

Binary data begins immediately after the final “t” in start. That is, there is no carriage return or linefeed in the file after “data_start”; there very next byte in the file is the first byte of the binary data.

At the end of the binary data, there will be the following string:

<CR><LF>data_end<CR><LF>

where <CR> is the ASCII carriage return character (0x0d, or decimal 13), and <LF> is the ASCII linefeed character (0x0a, or decimal 10). Note that <CR> and <LF> each denote a single byte in the file, so that the end marker flag "<CR><LF>data_end<CR><LF>" is 12 bytes long.

Description of individual file types

1. System setup files (".set")

The ".set" file is a master file which contains information relating to the hardware settings of the system, including things like channel gains and signal routing, filter configurations, tracker setup, etc. Everything that can be set in dacq2 in a way that persists between program runs will be stored in the ".set" file.

2. Tetrode/stereotrode data files (".1" to ".32")

Data from tetrode 1 (or stereotrodes 1 and 2) is stored in the ".1" file. Similarly, data from tetrode N is stored in file ".N", where N ranges up to 32. There are two general recording modes at present. The first is unit mode, and the second raw mode.

Unit mode

In unit mode, data is stored in 1 ms chunks, 200 µs and 800 µs after a threshold event. The header will contain a setting called "spike_format". Generally, there are 216 bytes per spike, structured as 54 bytes per channel, in order by channel number (so, for tetrode 1, the channels are in the order 1, 2, 3, 4, or actually 1a, 1b, 1c, 1d in dacq2 nomenclature, etc.). The 54 bytes consist of a 4 byte time stamp (most significant byte first), then 50 8-bit samples. The ASCII header specifies you how many spikes there should be in the file ("num_spikes"), and the "timebase" of the timestamps (almost always 96 kHz, which means the 4-byte timestamp value needs to be divided by 96000 to get a time in seconds). The samples themselves are normally collected at 48 kHz (again, specified in the header).

Tetrode and stereotrode data is almost identical, in that both store 4-channels worth of data per spike. The only difference is that in stereotrode mode, either channels 3&4 or 1&2 will be all zeroes, corresponding to a spike on stereotrode 1 or 2, respectively (the stereotrode format is very wasteful of file space).

Raw mode

In raw mode, data is stored continuously, rather than in spike packets. The header will contain a setting called "raw_format". It will also contain "num_samples", which specifies how many samples have been collected, and at which "sample_rate". Samples are normally 16-bit, but in any case this is specified by "bytes_per_sample" in the header.

3. Single electrode data files (“.spk”)

The “.spk” file is generated in single-electrode recording mode. Spike data is similar to the individual spikes of tetrodes or stereotrodes, in that it consists of 50 8-bit samples. However, after the data_start, the binary data blocks are 56 bytes long. The first two bytes indicate which electrode the spike was recorded on; the next four bytes are the timestamp; and the final 50 bytes are the spike waveform samples. The number of spikes is specified in the header by “num_spikes”.

4. EEG data files (“.eeg”, “.eegX”, “.egf”, “.egfX”)

EEG data is usually recorded continuously at 250 Hz in unit recording mode. The “.eeg” and “.eg2” files contain the data from the primary and secondary EEG channels, if these have been enabled. Very simply, the data consist of “num_EEG_samples” data bytes, following on from the data_start. The sample count is specified in the header. The “.egf” file is stored if a user selects a higher-sample rate EEG. Samples are normally collected at 4800 Hz (specified in the header), and are also normally 2 bytes long, rather than just 1.

5. Position files (“.pos”)

The format of the “.pos” file depends on the tracking mode. Each position sample is 20 bytes long, and consists of a 4-byte frame counter (incremented at around 50 Hz, according to the camera sync signal), and then 8 2-byte words. In four-spot mode, the 8 words are redx, redy, greenx, greeny, bluex, bluey, whitex, whitey. In two-spot mode, they are big_spotx, big_spoty, little_spotx, little_spoty, number_of_pixels_in_big_spot, number_of_pixels_in_little_spot, total_tracked_pixels, and the 8th word is unused. Each word is MSB-first. If a position wasn't tracked (e.g., the light was obscured), then the values for x and y will both be 0x3ff (= 1023). The header contains things like the limits of the tracking window in pixels, etc.

Note: the frame counter is NOT a timestamp for the position samples. It should be ignored in analyses. The samples are obtained at regular intervals so the X^{th} sample in the file was taken at time X/50 seconds since the start of the recording session.

6. Digital input/output and keypress timestamp files (“.inp”)

The “.inp” file contains timestamps for digital input or output and keypress events. The header specifies how many such events have been stored (“num_inp_samples”). Then, immediately following data_start there will be this many event blocks of 7 bytes, each defined as follows:

byte 1 = most-significant byte of timestamp
byte 2 = next-most significant byte
byte 3 = next-most significant byte
byte 4 = least-significant byte of timestamp

So, by read the four bytes and you get a 32-bit number. Divide this number by 1000 (or whatever the “timebase” is specified as in the header) and you get the time of the event in seconds from the beginning of the trial.

byte 5 = either "I" or "O" or "K"

"I" means digital input, "K" means keypress, and "O"" means .

If it is "I", then

byte 6 = value of digital input channels 16, 15, ..., 9
byte 7 = value of digital input channels 8, 7, ..., 1

i.e., the least significant bit corresponds to the smallest channel number.

If it is "O", then

byte 6 = value of digital output channels 16, 15, ..., 9
byte 7 = value of digital output channels 8, 7, ..., 1

i.e., the least significant bit corresponds to the smallest channel number.

If it is "K", then if byte 6 = 0, this implies that it was a normal key (not a function key like F1, etc.), and byte 7 = ASCII value of the pressed key.

If it is "K" but byte 6 <> 0, then this implies that it was a function key, and byte 6 contains the function key code. Byte 7 is undefined in this case. Function key codes frequently used include:

F1 - F10 = code 59 - 68
ShiftF1 - ShiftF10 = code 84 - 93
CtrlF1 - CtrlF10 = code 94 - 103
AltF1 - AltF10 = code 104 - 113

All of the other buttons on the keyboard generate codes as well. The can be found by searching the internet and looking for "IBM keyboard scan codes".

7. Stimulation pulse timestamp files (“.stm”)

If *dacqUSB* is set for stimulation during unit or raw recording modes, a “.stm” file will created that contains stimulus pulse times. The header contains the count of the number of timestamps stored (“num_stm_samples”), and immediately after the data_start, the file contains this many 4-byte timestamps. The timebase for the timestamps (usually 1 kHz) is stored in the header as well. Dividing each timestamp by the timebase gives the stimulus time in seconds from the start of the trial.

8. Raw data files (“.bin”)

The “.bin” files produced by *dacqUSB* are designed to be an intermediate state because the intention is that users should save their data into Tint or BPF format at the end of a trial. At present, it lacks a global header separate from the individual packet headers, and will probably get some simplifying improvements at some point.

Basically, a data.bin file consists of 432-byte binary packets. Each packet consists of a 32-byte header, and 384 bytes of electrode data, and then a 16 byte trailer.

Header

The header consists of:

4 bytes ID (will be "ADU1", unless the tracker position record is populated with valid data, in which case it will be "ADU2")

4 bytes packet number

2 bytes digital inputs

2 bytes sync inputs

20 bytes tracker position record (only valid data if packet ID is "ADU2") -- same format as standard .pos file position records.

Data

Then there are three samples x 64 channels x 16-bits (= 384 bytes), followed by 16 dummy bytes at the end to make up the total packet length of 432. The samples order is given below. Each sample is two bytes long, in 2's complement.

The data are stored at 48 kHz, so you should have 16000 packets of 432 bytes per second of recording. Yes, this is very inefficient because you don't have anywhere near 64 channels so it is mostly wasted space; this will be improved in a future version update.

The main complication is that the order of the channels in the data file is not something nice like 1,2,3, ... Instead, there is a remapping function:

```
remap_channels : array [1..64] of word = (
  32, 33, 34, 35, 36, 37, 38, 39,
  0, 1, 2, 3, 4, 5, 6, 7,
  40, 41, 42, 43, 44, 45, 46, 47,
  8, 9, 10, 11, 12, 13, 14, 15,
  48, 49, 50, 51, 52, 53, 54, 55,
  16, 17, 18, 19, 20, 21, 22, 23,
  56, 57, 58, 59, 60, 61, 62, 63,
  24, 25, 26, 27, 28, 29, 30, 31 );
```

For instance, if you want to find the data for channel 7, you look at remap_channels[7], which is 38. So, in the 432-byte packet, you ignore the 32 byte header, and the data for channel 7 will be at:

bytes 32(header)+(38*2), and 32+(38*2+1) (first sample low and high bytes)

bytes 32(header)+128(first samples, 64 ch x 2 bytes)+(38*2), and 32+128+(38*2+1) (2nd sample)

bytes 32+128+128+(38*2), and 32+128+128+(38*2+1) (third sample)

and so on.

Trailer

Finally, the trailer consists of 16 bytes:

2 bytes contain a record of digital output values

2 bytes contain stimulator status

10 bytes of zeroes (reserved for future use)

2 bytes contain the ASCII keycode if a key was pressed during the time the packet was active.

9. Field potential parameter files (“.epp”)

The “.epp” file contains field potential mode waveform parameters, as extracted in real-time during recording. In order to read the parameters file, it is necessary first to read from the header the values for “num_param_cols” and “num_waves”.

Immediately after the data_start string, the file will consist of “num_waves” records, each of which is 6 * “num_param_cols” bytes long. The records consist of 6-byte real numbers (this is called real48 in Delphi). The order of the “num_paral_cols” real values is the same as the order of the parameter names in the header “paramname_1”, “paramname_2”, etc.

For example, if the header contains this:

num_waves 39
num_param_cols 16
paramname_1 Slope V/s
paramname_2 EPLat ms
paramname_3 Spike mV
paramname_4 SpLat ms
paramname_5 Max mV
paramname_6 Min mV
paramname_7 Max - Min
paramname_8 Sp Area
paramname_9 Stim Artf
paramname_10 Slope Lev
paramname_11 Slope Lat
paramname_12 Slope End
paramname_13 Sp Start
paramname_14 Sp End
paramname_15 Resp End
paramname_16 Time ms

it would mean that after data_start there is:

6-byte real value Slope for wave 1
6-byte real value EP latency for wave 1
...
6-byte real value Time of response for wave 1
6-byte real value Slope for wave 2
6-byte real value EP latency for wave 2
...
6-byte real value Time of response for wave 2
...
6-byte real value Slope for wave 39
6-byte real value EP latency for wave 39
...
6-byte real value Time of response for wave 39

followed by the data_end string.

Note that the Delphi 6-byte real format is somewhat unusual. The bit pattern is consists of a sign bit, followed by 39 fraction bits, followed by 8 exponent bits:

SFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF EEEEEEEE

Conversion to IEEE754 floating point double precision is discussed here, and some helpful links are provided: http://rob.wemakewebsites.co.nz/index.php?p=40. A program to convert to C double is provided here: http://www.wotsit.org/list.asp?search=Borland+Pascal+Real&button=GO! Conversion to double for Java is shown here: http://siarp.de/node/191.

10. Field potential waveform files (".epw")

The ".epw" file contains field potential mode raw waveforms. The header contains the following relevant values: "num_waves", "bytes_per_timestamp", "samples_per_wave", and "bytes_per_sample".

After the data_start string, the file will consist of "num_waves" records, each of which is of size "bytes_per_timestamp" + ("samples_per_wave" * "bytes_per_sample").

For example, if the header contains these values:

num_waves 39
bytes_per_timestamp 4
samples_per_wave 1920
bytes_per_sample 1

it would mean that after data_start there is:

4-byte timestamp for wave 1
1920 * 1-byte samples for wave 1
4-byte timestamp for wave 2
1920 * 1-byte samples for wave 2
...

4-byte timestamp for wave 39
1920 * 1-byte samples for wave 39

followed by the data_end string.

11. Files produced by scripts (e.g., “.log”)

A DACQBASIC script running during a trial can create a file called something like “data.log” into which the script can write anything (e.g., choices the subject has made during a trial processed in real-time according to some criteria). This file will be stored, at the end of the recording session, along with all standard data files, to the user-designed dataset name like “trial1.log”. The format of the file is a function of the script that creates it, but it will invariably consist of ASCII text, because DACQBASIC is unable to create binary output files.