Deuteron Technologies Ltd

Electronics for Neuroscience

Data File Reference Manual

Published: 1/22/2020

Table Of Contents

1 Overview

This reference manual is intended to provide full documentation of all of the types of data files that are generated by Deuteron Technologies' family of neural and audio data loggers. It contains both basic explanations of the file types presently in use and exact technical details of the structure of each type of file.

The most detailed information provided in this manual may be needed by users who wish to analyze the recorded data by writing their own programs to access the recorded data files directly. In many cases, direct access of the recorded files is very easy, requiring, for example, a single MATLAB command to move data from a recorded file to MATLAB's internal format. Other users may choose less direct methods to access the recorded data, in which case they may not need all of the technical detail provided, but may still wish to read the general explanations provided in the first sections of this manual. Alternatives to writing programs that directly access the data files include:

- Using stand-alone data viewer programs provided by Deuteron Technologies
- Writing programs that use function libraries provided by Deuteron to access the data
- Using existing data analysis programs written by colleagues and others
- Using utilities that translate Deuteron's files to other formats for which analysis tools exist

This manual covers two basic types of data files: The simple "Flat" format used exclusively until 2019, and the "Block" format introduced in September 2019. Some devices will continue to Flat format well after 2019.

This scope of this manual is limited to the description of the structure of the recorded data files. It does not cover general instructions for using Deuteron's neural recorders, which are provided in the instrument's instruction manual. It also does not cover methods for analysis and feature extraction of the recorded data.

The devices that generate data files described in this manual include those shown in the table below.

Device names	Number of neural channels	Motion sensor	Audio	Notes
Audio logger AL1	None	9-axis **	Ultrasonic	
Audio logger AL2	None	9-axis **	Ultrasonic	
MouseLog16B MouseLog16V	16	None	None	Single-board
MouseLog16C	16	9-axis **	Ultrasonic **	Single-board, smallest
MouseLog16F, MouseLog16LP	8 selected of 16	None	None	Low power for LFP, EEG, EMG
Ratlog64 series	32 or 64	9-axis	Optional, audible	Modular
RatLog128	32, 64 or 128	9-axis	Optional	Modular
SpikeLog64	64	9-axis **	Ultrasonic **	Single board
SpikeLog128_1 SpikeLog128CP	128	9-axis **	None	Single board with adapters
SpikeLog16	16	9-Axis	None	Formerly called NeuroLog16, larger, discontinued

Note: Features marked ** are only available when using software versions that support "Block" format.

2 General information about Deuteron’s recorded data files

Neural data sets can generate very large amounts of unprocessed data, for example a 2-hour, 64-channel neural recording session generates about 28GB of unprocessed data. Manipulating such data sets as a single data file is generally impractical; besides, some computers cannot accept files larger than 4GB.

All of Deuteron’s devices record data by generating sequences of files of fixed length. Each file is exactly 16MB (16777216 bytes) long. A typical device will generate a new file every few seconds. Having many files of fixed length has several advantages both when generating the files and when later transferring and processing them.

When a recording is started, a new file is started. When a recording is stopped, this will usually happen in the middle of a file. The last file of a recording will still have the same length as all the other files, and the unused space at the end of the file will be blank.

2.1 Preventing a computer from writing to the memory card

Data recorded using Deuteron’s loggers can be read by any almost any computer and any operating system. However, as explained in the device’s instruction manual, you should copy the data from the logger’s memory card to a computer and only then process the data. You should not re-name or erase the data on the memory card using a computer; you must erase the data on the memory card using only the logger. Similarly, you should not add any data files to the logger’s memory card using a computer. The best way to prevent accidental writes by the PC to the logger’s memory card is to us an SD card adapter. This is an adapter that is the size of a full size SD memory card (32mm x 24mm) into which one can insert the logger’s micro-SD card. Each logger is provided with several of these adapters. Accidental writes by the computer can be completely prevented by ensuring that the write-protect switch of the adapter is permanently in the “read-only” position, that is, with its tab closer to the center of the edge of the adapter.

If data is accidentally written by a computer to a logger’s memory card, you should follow the instructions given in the device’s instruction manual for restoring the card’s file system before attempting to use the card to log more data.

2.2 Selection and preparation of memory cards

Deuteron’s loggers use ordinary commercial MicroSD cards. Not all MicroSD cards are the same, and some are much better suited for use with Deuteron’s loggers than others. If you wish to use a memory card other than that supplied with the logger, you can do so, but you must ensure that the card is of a suitable type and that it has been formatted and configured according to the instructions provided in the logger’s instruction manual. Deuteron periodically recommends particular brands and models of memory cards that have been qualified and tested with a particular kind of logger. There are two main criteria for a card being suitable for use with a given logger. The most important criterion is that the card has suitable write-time statistics. The time taken to write a block of data to an SD card is not constant. If a card takes too long to write a block of data, that block might be lost. If too many blocks of data are lost, the logger will stop recording and report an error. A suitable card should not lose any data. For more information about dropped blocks, see here (link). The other criterion is power consumption. Some memory cards consume much more power than others, so for a given battery size, an unsuitable card may provide a much shorter recording time than another.

Before a new memory card is used with a Deuteron logger, it must be correctly formatted and configured. Instructions for doing this are provided in the logger’s instruction manual.

3 Introduction to “Flat” and “Block” file formats

This section relates to files of recorded neural or audio data. For event log files, so a later section.

3.1 *Flat file format*

Until September 2019, all of Deuteron’s data loggers used a simple arrangement of data within its files, called “Flat” data format.

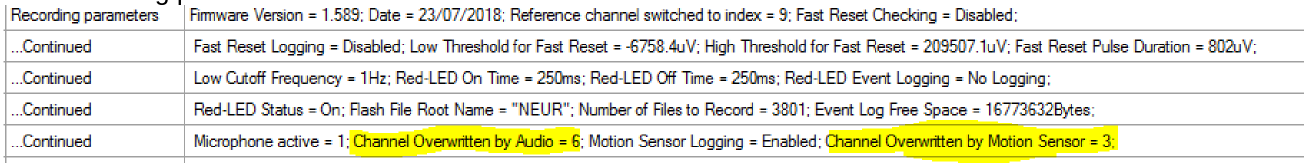
The Flat data format is essentially an array of 2-byte integers of unprocessed data, such that each 16MB data file contains 8 388 608 (= 2^{23}) integers, each representing a digitized sample of incoming data, recorded in the order that the sample was made. For example, if the file represents a recording of 32 channels of data, the file can be regarded as having 32 columns of data, one column for each channel of recorded data, and each file containing 262144 (= 2^{18}) rows, each row representing one sample period. This data format is most convenient where there is only one major source of recorded data and where the number of channels is an integer power of 2. This format is very easy to process; for example a single “fread” command in MATLAB can read such file into a MATLAB array.

When Flat file format is used, synchronization events are stored in a single separate file dedicated to logging of such events together with their respective timestamps, called an event log file. Details of event log files are provided in a later section.

Likewise, a Flat data file contains no “metadata”, that is, information about the context or structure of the recorded data, such as the sampling rate, the number of channels, the timing of the data or the various settings of the system. When Flat file format is used, metadata is stored in the event log file, and the number of channels is used to determine the extension part of the data file’s name (e.g. DT2, DT4) as detailed later.

3.1.1 Flat file format containing more than one type of data

The Flat file format is sometimes used with more than one data source, meaning it may be possible to record neural and/or motion sensor and/or audio data simultaneously. For example, the RatLog64 logger can record data from its motion sensor by replacing a user-specified column of its neural data with data from the motion sensor. This has the disadvantage of requiring the user to forfeit one column (user-specified) of recorded neural data. Audio data is stored similarly to neural data; it is recorded at the same frequency as neural data, and each data point of audio data is stored in a 16-bit integer. Thus, the timestamp of each audio data point is the same as the timestamp of the neural data in the same row. The motion sensor data are recorded at a different frequency and have a unique storage structure, the exact details of which are provided later in section 5.4. Which channel(s) alternate types of data have been stored in can be found in the event log file in the Recording parameters event.



Recording parameters	Firmware Version = 1.589; Date = 23/07/2018; Reference channel switched to index = 9; Fast Reset Checking = Disabled;
...Continued	Fast Reset Logging = Disabled; Low Threshold for Fast Reset = -6758.4uV; High Threshold for Fast Reset = 209507.1uV; Fast Reset Pulse Duration = 802uV;
...Continued	Low Cutoff Frequency = 1Hz; Red-LED On Time = 250ms; Red-LED Off Time = 250ms; Red-LED Event Logging = No Logging;
...Continued	Red-LED Status = On; Flash File Root Name = "NEUR"; Number of Files to Record = 3801; Event Log Free Space = 16773632Bytes;
...Continued	Microphone active = 1; Channel Overwritten by Audio = 6; Motion Sensor Logging = Enabled; Channel Overwritten by Motion Sensor = 3;

3.2 *Block File Format*

3.2.1 Aims of Block file format

Deuteron introduced its newer “Block” file format in September 2019 to support loggers that have more than one significant data source. For example SpileLog64 records 64 channels of neural data, one channel of ultrasonic audio data and data from its 9-axis motion sensor. The main requirements of the Block file system are:

- The ability to store data from up to 7 different types of signal sources that may have independent sampling times
- The ability to efficiently store neural data with any number of neural channels, not necessarily an integer power of 2
- The integration of synchronization event data with streamed neural or audio data in one file
- Sufficient metadata in each data file such that any file can be independently analyzed without reference to other files
- The ability to write data efficiently to a memory card as required in the context of a low-power logger
- Optimal use of storage capacity of the memory card
- Provision for storing data generated by undefined future features

Presently supported types of signal source are

- Digitized neural data
- Digitized audio data
- Motion sensor data
- Synchronization events
- RTK data

Signal sources that will be supported in future include:

- Data from external digital devices connected to the logger
- High-speed thermometry data for study of breathing and sniffing
- Compressed video signals from an animal-borne camera

3.2.2 Outline of structure of Block file format

The Deuteron Block File Format recordings stored on the Micro-SD memory card include two types of files:

a) General data files that contain neural, audio, motion-sensor data as well as synchronization event data.
b) Dedicated event Log files that contain only synchronization event data.

Unlike in the flat file format, events that occur ***during a recording*** appear in the data files along with other types of data, thus dedicated event log files contain *only* events that occur ***between*** recordings. Each file is a constant size of 16MB, and files of each type (data vs event log files) are generated sequentially.

Each data file contains event data and may also contain any combination of data from neural, audio, motion or other sensors. To efficiently store multiple types of data in an organized fashion, each file is divided into blocks containing all data recorded during a particular time period. These blocks are of a predetermined size; at the time of this writing, all block format files use a fixed block size of 64KB (65536 bytes) and contain 256 such blocks. Each block begins with a header containing a timestamp along with meta-data describing how the block is structured. This header contains information concerning the placement and size of each data type section within the block. The time period spanned by a block depends on the types of data being recorded, and thus may vary between recordings.

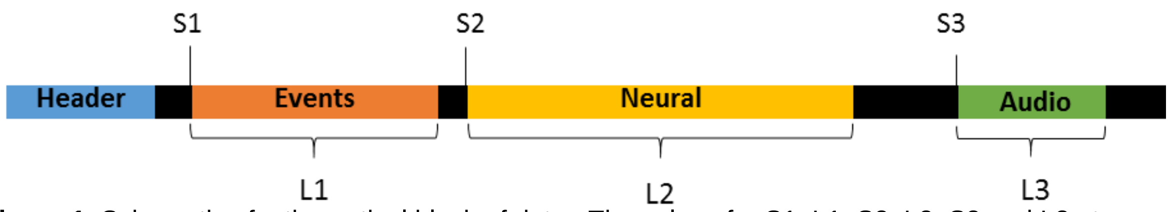
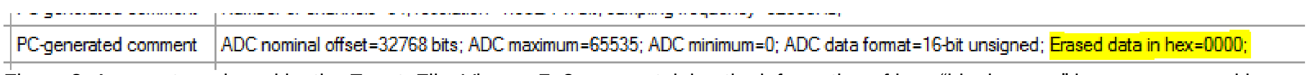


Figure 1. Schematic of a theoretical block of data. The values for S1, L1, S2, L2, S3 and L3 etc. are found in the header. Note that while every file contains block headers and event data, the presence of other data types depends on the selections made by the user when starting the recording.

Each recording session starts at the beginning of a file. When a user stops a recording session, the remainder of the file being recorded is left blank. Blank space on a memory card is generally filled with zeros. However, some in some brands of SD cards, notably Samsung blank space is filled with ones, (e.g. 0xFFFF). The expected state of blank space for a given memory card can be found in the Events section at the beginning of each recording as one of the PC-generated comments having the key “Erased data in hex”, as shown in the excerpt below:



PC-generated comment | ADC nominal offset=32768 bits; ADC maximum=65535; ADC minimum=0; ADC data format=16-bit unsigned; Erased data in hex=0000;

Figure 2. An event as viewed by the Event_File_Viewer_7_2.exe containing the information of how “blank space” in a memory card is stored.

3.2.3 Event log files

As mentioned above, event log files exclusively contain events that occur between recordings, as all other events are recorded alongside the concurrent data. Event log files are exactly 16MB long and are named EVENTnnn.DF1 where nnn is a three-digit integer value indicating the file number in chronological order starting with 000.
Event log files are formatted in an identical manner to data files (i.e. they are divided into blocks with the same header as data files have), with the exception that they will contain event data exclusively. They can be viewed or retrieved alongside events that occurred during recordings using the Event_File_Reader_7_2 executable or DLL. Details on how to use these tools with programming examples can be found in the document Event_File_Viewer_7_2_Manual.

4 Overview of tools

For loading or viewing events, you can use the Event_File_Viewer_7_2.dll or exe respectively. It accepts files of both flat and block file format that contain events (i.e. files in the flat format with the NLE extension and any file in the block format). The executable will run as an application that allows you to view, sort, and filter events by event type. The DLL has example scripts that demonstrate how to run it from Matlab, C++, C#, Python, and Delphi. It loads events as strings into the memory. More detailed instructions on how to use the Event_File_Reader_7_2 can be found in the instruction manual Event_File_Viewer_7_2_Manual.
In block file format, there may be many files containing events that you wish to view simultaneously. For more than ~10 files, we recommend compressing the events you wish to view into a single event file using the Event_File_Compressor.exe and then using the Event_File_Viewer_7_2 to process this output file.
For viewing data, we provide you with the DataFileViewer.exe. It works with both flat and block file format, and allows you to custom filter your data, view motion sensor, extract spikes, and view and play audio data. More details on the DataFileViewer.exe are contained in the DataFileViewer manual.
Furthermore, there are Matlab scripts available on the website that demonstrate data extraction and conversion to physical units from both flat and block file formats (excluding event data).

5 Detailed structure of Data files

The recorded data are stored on the Micro-SD memory card. In order to make this large data set more manageable, the recorded data are divided into files of exactly 16MB (=16777216 bytes = 2^{24} bytes). The data in each file are further divided into blocks of a constant size (currently 64kB), with each block containing a header followed by partitions, each of which contains data of a particular type recorded during a fixed time period. The header contains meta-data (i.e. information about the data), including the structure of the block, the types of data that are present and where data of each type is stored.
The header is 108 bytes long and it can be identified by its first 8-bytes which contain the constant identifier 0x1234ABCD 567890EF. The exact structure of the data in the header and what information it contains can be found in the table below:

Bytes	**Item**	**Description**
0…7	Constant identifier	The 64-bit constant 0x1234ABCD 567890EF (this is a number that should not exist in neural, motion or audio data).
8…11	File format ID	Currently this number should be a 1.
12…15	Block size	The size (in bytes) of a fundamental block (from the beginning of metadata to the end of neural data). This will typically be 65536 (=64kB), but can be larger or smaller.
16...19	timestamp	Time in ms from midnight (each block represents an integer number of ms)
20…23	Reserved	

24...107	Partition information	This is an array of 7 structures of 12 bytes each that defines how the data in this block is partitioned. Each block can have up to 7 different partitions (for 7 distinct types of data). Each 12-byte structure in the array is constructed as follows: 4 bytes defining the type of data, followed by 4 bytes defining the start position of the partition within the block, ending with 4 bytes defining the size of the partition in bytes. In C syntax, a single structure is defined as follows:

```
Struct
{
  uint32 Data_Type;        //enumerated types defined below
  uint32 Partition_Start;  //offset of first byte from the
                           //beginning of the block
  uint32 Partition_Size;   //range defined below
}Partition_Entry;__packed__
```

The types of data are as follows (we may add more types):

Index	**Data type**
0	No data
1	Event data
2	Neural data
3	Motion sensor
4	Audio
5	Reserved
6	Reserved
7 etc	Reserved

The partitions can be any size up to
<Block size> - <Block header size>.

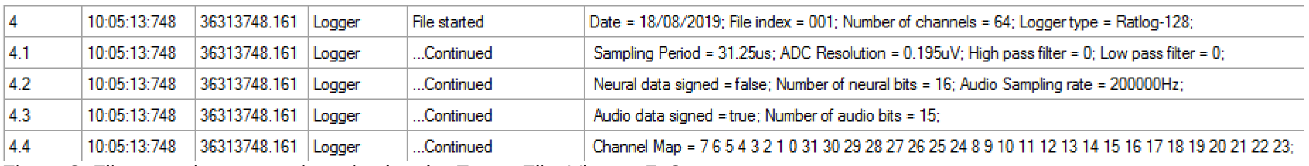
Table 1. Set up of a block header in bytes.

Based on the information contained in the “Partition information” section of the header, the user can find the data types present in the block, along with the location and size of the partition containing this data. Based on this information, the user can extract and concatenate the relevant data from each block to form complete data sets sorted by data type.

The file names used for the data files will be created with names in uppercase 8.3 format. They will have the format AAAAnnnn.DF1, where AAAA are any four uppercase letters or numbers, which can be set in the Basic Controls tab in LoggerCommand, and nnnn is 4-digit number assigned in order by the logger, the first being 0000 and the last being 3799 for a 64GB card .

5.1 *Event Data*

Similar to event log files, event data within regular data files can be viewed or retrieved using the Event File Reader executable or DLL. The executable and DLL extract the event data from the files. Details on how to use these tools can be found in the document Event_File_Viewer_7_2_Manual. Some events such as the file start event contain important information about the data such as sampling frequency, resolution, number of bits etc.



4	10:05:13:748	36313748.161	Logger	File started	Date = 18/08/2019; File index = 001; Number of channels = 64; Logger type = Ratlog-128;
4.1	10:05:13:748	36313748.161	Logger	...Continued	Sampling Period = 31.25us; ADC Resolution = 0.195uV; High pass filter = 0; Low pass filter = 0;
4.2	10:05:13:748	36313748.161	Logger	...Continued	Neural data signed = false; Number of neural bits = 16; Audio Sampling rate = 200000Hz;
4.3	10:05:13:748	36313748.161	Logger	...Continued	Audio data signed = true; Number of audio bits = 15;
4.4	10:05:13:748	36313748.161	Logger	...Continued	Channel Map = 7 6 5 4 3 2 1 0 31 30 29 28 27 26 25 24 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23;

Figure 3. File started event as viewed using the Event_File_Viewer_7_2.exe

5.2 *Neural data*

The Flat data format is essentially an array of 2-byte integers of unprocessed data, such that each 16MB data file contains 8 388 608 (= 2^{23}) integers, each representing a digitized sample of incoming data, recorded in the order that the sample was made. For example, if the file represents a recording of 32 channels of data, the file can be regarded as having 32 columns of data, one column for each channel of recorded data, and each file containing 262144 (= 2^{18}) rows, each row representing one sample period.

The neural data in each file is stored as an array of 16-bit unsigned integers, each representing a digitized sample of incoming data. They are stored in the same order as the samples are made. For example, in a 32 channel system, the first 32 16-bit integers stored correspond to the first data point of each channel.
The integers are stored in little-endian format, that is, the least significant byte comes first. To convert from this unsigned 16-bit value to volts, you will need to get the number of neural bits and the ADC resolution from the File started event and use the following formula:

volts = ADC resolution * (unsigned 16 bit value – 2^(number of neural bits – 1))

where ADC resolution and the number of neural bits can be found in the file started event (see figure 3 above)

The user can calculate the timestamp of any data point by extrapolating from the timestamp in the first block header in the file as follows:

$t_n = t_0 /1000 + \text{SAMPP} * n$

where t_n is the time in seconds from midnight of data point n, t_0 is the timestamp in the first block header in the file, and SAMPP is the sampling period. These values are found in the “file-started” event record (as in figure 3, for example). Data point n refers to the nth data point of a given channel. The voltage values range from approximately ±6mV.

5.3 Audio data

Audio data are stored as 16-bit integers though they may only be 14 or 15 bit values, and they can be signed or unsigned. The “file-started” event record in the events section of the first block of each file contains information needed to convert the integer data to physical units. This includes

- The number of audio bits
- Whether the data are signed or unsigned
- The sampling frequency
- Amplification settings / audio resolution

To convert the integers to physical units, use the following formula:

Audio_data = integer_value * Audio_resolution.

The audio resolution in micropascals per bit (µPa) is provided as follows:
High gain: 60 µPa
Low gain: 400 µPa
Note these values are nominal and subject to broad tolerance.

The time of any given sample =
$t_n = t_0 + n / \text{SamplingFrequency\_Hz}$

where t_n is the time midnight of data point n, t_0 is the timestamp in the first block header in the file. Note that the timestamp t_0 is provided in milliseconds. SamplingFrequency is a value found in the “file-started” event record.

5.4 Motion sensor data

All motion-sensor data are stored as 2-byte integers. The three accelerometers and the three gyroscope axes of the motion sensor are each sampled at 1kHz, while the three magnetometer axes are sampled at approximately 111Hz, that is every 9ms. Magnetometer data are logged at a rate of 1kHz but measurements are only taken once every 9ms, so approximately 9 adjacent magnetometer values will be repeated values.

Embedded in the block of motion sensor data are metadata (data about the data) that define the exact number of samples of each kind in each block, and the location of the data relative to the beginning of the block. Also included is a time-stamp which signals the start time of the block.

Since motion sensor data in a given block is written with a time lag compared to other data in the block, it is critical to use the motion sensor's time stamp (rather than the block's general timestamp) when analyzing motion sensor data.

The timestamp for motion sensor data is the time of day expressed as the number of seconds since midnight multiplied by 16000. The timing resolution is thus 62.5µs. To obtain the time of any motion-sensor value, use this formula:

$t_n = t_0 + n$ / SamplingFrequency_Hz

where t_0 (in seconds) is the timestamp of the motion sensor record (see below) / 16000 and n is the index of the sample in the block (i.e. within a single axis).

5.4.1 Format of the Motion Sensor Record

The record is an array of 16-bit words, which is structured as follows:

Word offset	**Description**	**Notes**
0	Constant identifer [0]	= 13579 Decimal
1	Constant identifer [1]	= 24680 Decimal
2	Word offset of start of accelerometer data (accl_data_start)	This constant describes the position in the record of the accelerometer data. (Determined based on system parameters.)
3	Word offset of start of gyroscope data (gyro_data_start)	This constant describes the position in the record of the gyroscope data. (Determined based on system parameters.)
4	Word offset of start of magnetometer data (mag_data_start)	This constant describes the position in the record of the magnetometer data. (Determined based on system parameters.)
5	Reserved	The constant 0
6	Number of words of valid accelerometer data in this record	
7	Number of words of valid gyroscope data in this record	
8	Number of words of valid magnetometer data in this record	
9	Reserved	The constant 0
10-11	Timestamp	msec since midnight * 16
accl_data_start	Start of accelerometer data	the offset (in 2-byte words) of the first byte of accelerometer data from the beginning of the motion sensor section of the block
gyro_data_start	Start of gyroscope data	the offset (in 2-byte words) of the first byte of gyroscope data from the beginning of the motion sensor section of the block
mag_data_start	Start of magnetometer data	the offset (in 2-byte words) of the first byte of magnetometer data from the beginning of the motion sensor section of the block

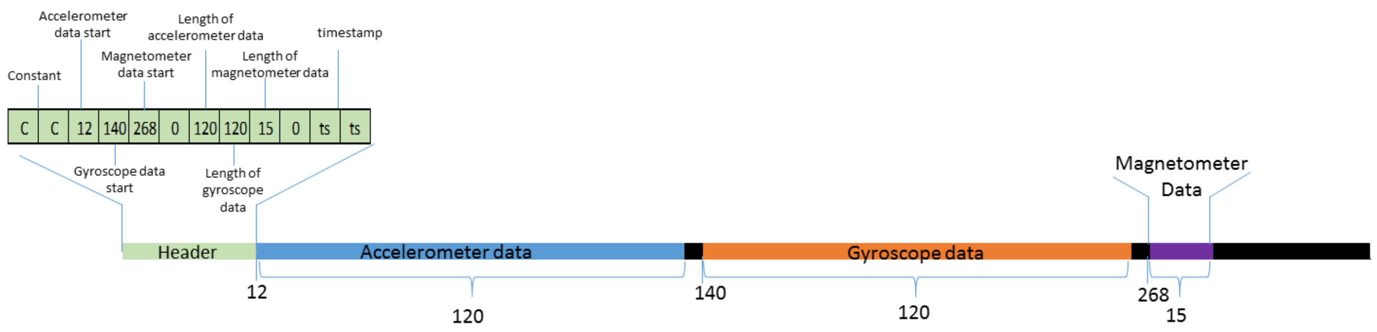


Figure 4: The structure of motion sensor data in all file formats.

Within each segment of type-specific motion sensor data (e.g. accelerometer data in a single block), each data point contains a value corresponding to the x, y, and z axis, stored in that order. For example, the first 3 values in the accelerometer data segment in a block correspond to the x, y and z values (respectively) of the first accelerometer data point of the block.

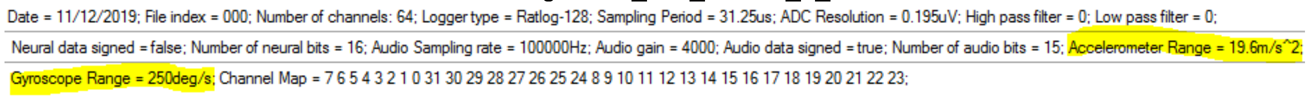
5.4.2 Converting motion sensor data from integers to physical units

To convert the motion sensor values to physical units, you must know the number of bits used to store this data type and the maximum value that can be stored. These values may be fixed, hardware dependent, or set by the user. They can be found in the table below:

	Number of bits	**Maximum value**
Accelerometer	16	User defined (see below)
Gyroscope	16	User defined (see below)
Magnetometer	**Spikelog16, Ratlog64:** 13 **Other:** 14	**Spikelog16, Ratlog64:** 1200 μTesla **Other:** 4800 μTesla

Table 2: Motion sensor meta-data

The maximum accelerometer and gyroscope values are stored in the file started event in the event section of the file. This information can be obtained using Event_File_Viewer_7_2.



```
Date = 11/12/2019; File index = 000; Number of channels: 64; Logger type = Ratlog-128; Sampling Period = 31.25us; ADC Resolution = 0.195uV; High pass filter = 0; Low pass filter = 0;
Neural data signed = false; Number of neural bits = 16; Audio Sampling rate = 100000Hz; Audio gain = 4000; Audio data signed = true; Number of audio bits = 15; Accelerometer Range = 19.6m/s^2;
Gyroscope Range = 250deg/s; Channel Map = 7 6 5 4 3 2 1 0 31 30 29 28 27 26 25 24 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23;
```

Figure 5: The maximum accelerometer and gyroscope values for this recording as viewed with Event_File_Reader_7_2.exe.

Each data point is then scaled as follows:

$s = v * \text{maximum value} / 2^{(\text{number of bits} - 1)}$

where s is the scaled data point (in physical units), v is the original data point, maximum value is found in the event in Figure 5, and number of bits can be found in the above table (Table 2).