Deuteron Technologies Ltd
Event File Viewer 7.2

Author: Stefanie Glowinsky

Date: 26/01/2020

Contents

1 Introduction

Deuteron loggers can produce files in two different formats; block file format and flat file format. In flat file format, data files (with extensions ".DT2", ".DT4", ".DT6", ".DT8", or ".DAT") store continuously measured (e.g. neural, audio, motion sensor) data, whereas the special event log file (".NLE") stores one-time events which contain metadata (information about the data), and timestamp and description of each event that occurred. In block file format, events are stored in files alongside continuously recorded data. These files are called "AAAAXXXX.DF1" where AAAA is a 4 letter prefix, and XXXX is the chronological index of the file starting at 0000. Events occurring outside of a recording are stored in files that contain events exclusively, called EVENTXXX.DF1, where XXX is the 3 digit index of the file, starting with 000.

Events can signify many things such as the beginning or end of a recording, LED lights turning on/off, or a stimulus being delivered to the animal. The Event File Reader 7.2 is a tool for parsing, viewing, and saving the event data recorded by data loggers in a convenient and easy to use manner. It can be launched by clicking on the executable to open the user interface, or from the MATLAB command line which can be used to either open the user interface or save the data directly to a CSV file.

Furthermore, information can be retrieved from an event log file using the Event_File_Reader_7_2.dll. Examples showing how to use the DLL are available in Matlab, Python, C++, and C#.

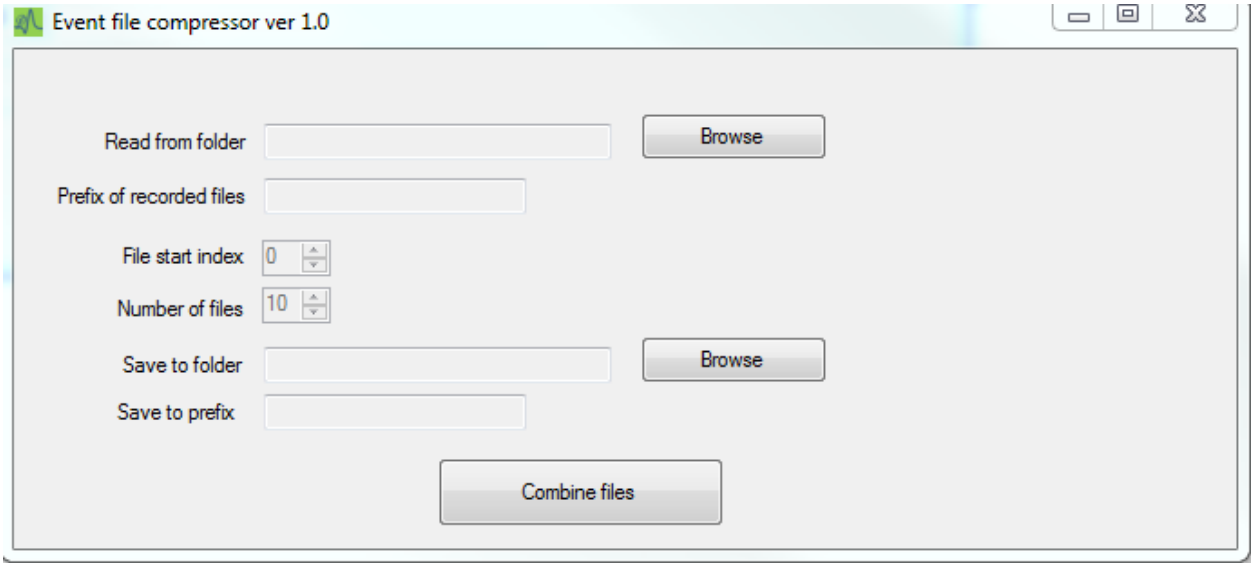
2 Getting started

2.1 Setup

Place the executable and the Settings.ini file in the same directory. Do not directly modify the Settings.ini file or change its name, as this may corrupt it or render the program unable to find it, preventing the program from running correctly.

2.2 Handling a large number of event files

In block file format, events are distributed throughout the files of a recording. To view events from many files at once (>10), we recommend you first compress them into a single event file using the EventFileCompressor.exe application.



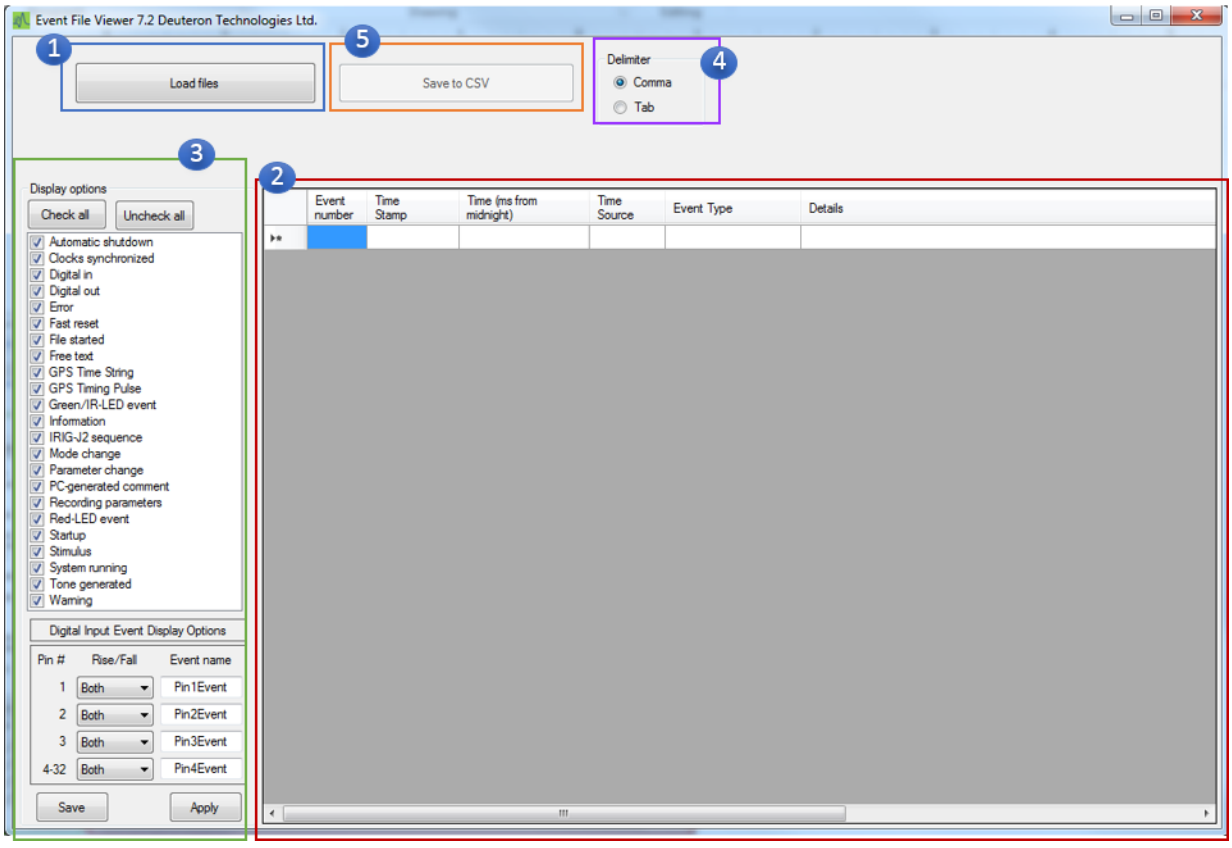
To use the event file compressor, you must browse for the folder containing the files you wish to compress. If there is more than one set of files in your chosen folder (i.e. different 4 letter prefixes), a dialog box will ask you which recording you would like to compress. You can then choose the first file and the number of files to be included. You may then enter where you would like the new compressed file to be saved and you may assign it a 4 letter prefix. When you click “Combine files”, the files will be compressed into a single file which you may then view or load using Event_File_Reader_7_2.

2.3 Running Event File Reader 7.2

The Event File Reader can be launched directly or from the MATLAB command window. Instructions for launching the event file reader using the command window can be found in section 4.
Double clicking on the executable will launch the user interface without initially displaying any files. The user can then choose one (flat format) or many (block format) files to view using the interface. You can also open single event log files directly with the event file viewer by setting Event_File_Reader_7_2 as the default program for opening it and then double clicking on the desired event log file.

3 User Interface

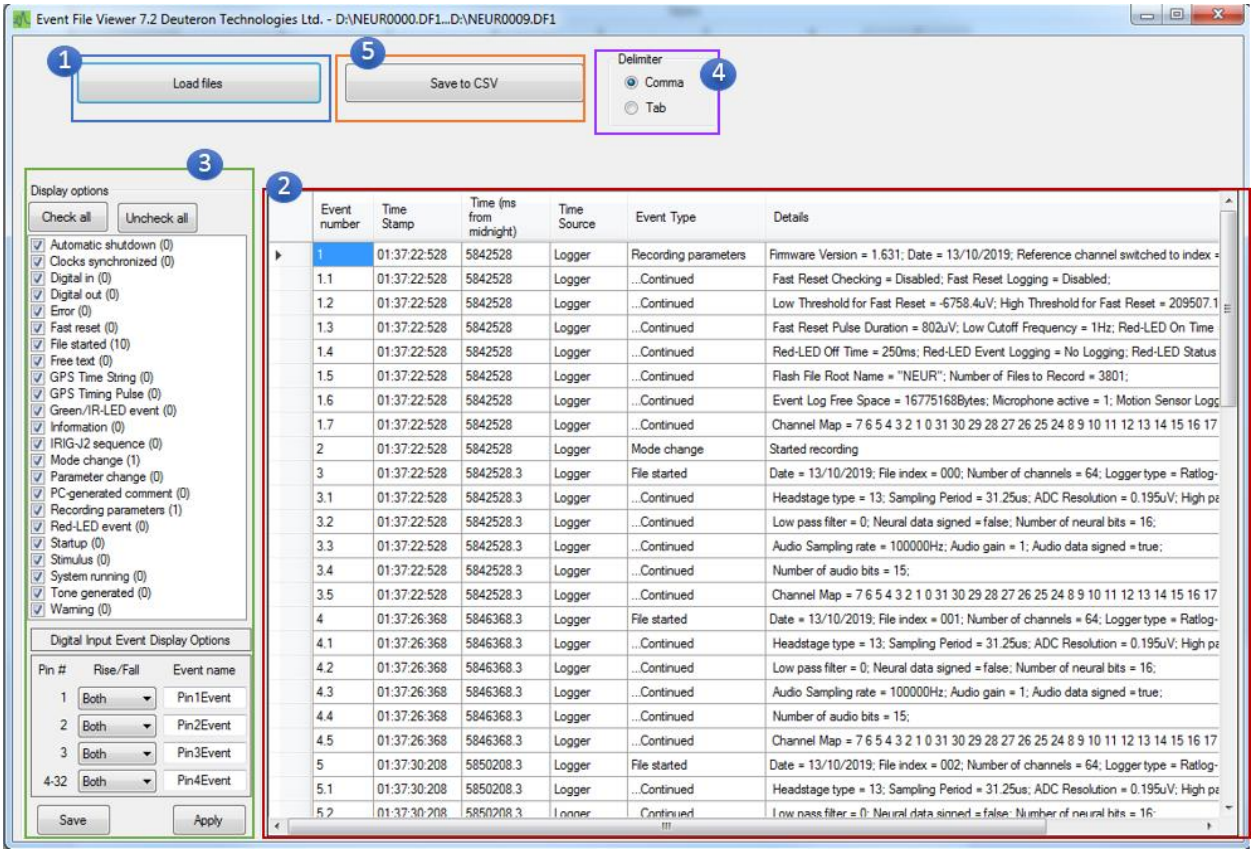
3.1 User Interface overview



Above is a first look at the GUI before loading an event file. Each of the following controls will be described in greater detail in the following sections:

1. Load file(s)
2. The data table where the event log file data are displayed
3. Controls to choose which kinds of events should be displayed and/or saved to the CSV file
4. Which delimiter should be used for value separation when writing to a CSV file
5. Save data to CSV. Button becomes enabled when a file is loaded.

3.2 Loading Event Log File(s)



Clicking on the “Load files” button (the button labeled 1 in the above image) will open a file dialog that allows the user to browse for file(s) containing events to load. Upon clicking “OK” after selecting a set of files, the events from those files will immediately be displayed in the table (labelled 2 above). Once the files have been loaded into the program, the “Save to CSV” button (labelled 5) will become enabled so the user can save the data from these files.

3.3 Data table

The data table is the table displaying information about the events found in the chosen files (labeled 2 in the above figure). For files in the flat file format, some general information about the file such as the firmware version and the date and time of the creation of the file, will appear just above the data table. For block file format, this information will be contained in the events in the table.
The data table itself consists of 6 columns which each address different aspects of the event:

Event number – the index of the event in the order in which the events were recorded to the file. When an event traverses multiple lines, the event number resorts to decimals. In the above example, the Recording parameters event (Event number 1) continues on for 7 additional lines which are labeled 1.1 – 1.7.

Time Stamp – gives the time stamp of the event as HH:MM:SS.mmm.

Time (ms from midnight) – the time stamp of the event as milliseconds from midnight. Some event types are measured with sub microsecond resolution while others have only millisecond resolution.

Time Source – If the time of this event was measured according to the logger, it will display “Logger” (in the case of millisecond resolution) or “Logger (Fine)” (sub microsecond resolution). Similarly, if the time of the event was measured according to the transceiver, it will display “Transceiver” (in the case of millisecond resolution) or “Transceiver (Fine)” (microsecond resolution).

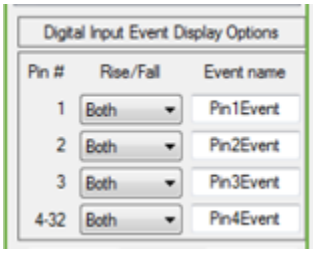
Event Type – there are many different kinds of events. For a description of each type of event, refer to Appendix A.

Detail – description of what has actually occurred in the given event.

3.4 Display control panel

The display control panel on the left hand side allows the user to choose which types of events should be displayed in the table and/or written to the CSV file.

The checklist contains a list of event types, and each event type name is followed by the number of times it occurs in the current file in brackets. The user can check or uncheck these options individually, or as a group using the check/uncheck all buttons.



The “Digital Input Event Display Options” section (above figure) allows the user to control in greater detail which digital input events should be displayed and how. For each pin (1, 2, 3, or 4+), the user can select from the drop down options whether they would like to display events signifying a digital rising edge, falling edge, both, or neither. The user can also use the text boxes in the “Event name” column to enter a custom name for the events occurring on each pin.

In order for the user’s chosen settings to be reflected in the Data table, the user must click “Apply” in the bottom right corner of the display control panel. The “Save” button in the bottom left corner of the display control panel will save the current settings and automatically apply them next time the Event File Reader program is run.

Note: Whereas the “Digital Input Event Display Options” allows the user to choose which pin events to display by pin number and rise/fall status, the “Digital in” checkbox will display, if available, the digital input port status and the digital input event status which is simply an alternative mode of displaying the

digital input events. For more information on the different ways of representing the digital input events and how to interpret them, see Appendix A.

3.5 Delimiter Control

The delimiter (labelled 4) control allows the user to choose the type of delimiter that will be used to separate values in the CSV file to which the data are saved. The options are either comma delimited or tab delimited.

3.6 Save to CSV

Clicking on the "Save to CSV" button (labelled 5) will open up a save file dialog, allowing the user to choose a location and name for the CSV file that will be created. Only events that are marked to be displayed will be recorded in the CSV file. To record all events, ensure that you first click "check all" and "apply" in the display options tab.

4 Command line version

4.1 Running the command line version from MATLAB

The event file viewer can be run from the command line in MATLAB.
To do so, the user should employ the system command. The arguments are as follows:
1. The name of the executable. (required)
2. **Flat file format:** The name of the .NLE event log file from which to read. This must end in the extension ".NLE". (optional)
Block file format: The name(s) of the ".DF1" files from which to read. If there are multiple files, they must contain the same 4 letter prefix and be continuous (e.g. NEUR0000 must be followed by NEUR0001). Each file name should be separated by a space.
3. The name of the .CSV file to write to. This must end in the extension ".CSV". (optional)
4. The type of delimiter. The default is comma, and this will be used in cases where the user enters only 3 inputs. To use tab as a delimiter instead of the default comma, type "tab" as the fourth argument. (optional)

For inputs 1-3, if they are not in the current file directory they must be entered with the full path.

If the user does not enter the name of a csv file to write to, the executable will open as a GUI. Otherwise the executable will run without opening the UI, read the files, and write the events to the requested .CSV file using commas as delimiters. If the user enters the final argument as "tab", the program will parse the event log file and write it to the requested CSV file using tabs as a delimiter.

Please note: the total length of the input arguments must not exceed 256 characters.

4.2 MATLAB Examples

In the following examples, the current directory is the location of the executable, the file to read from, and the file to write to:

```
% to open the GUI
s = system('Event_File_Reader_7_2.exe') ;
```

```
% to open the GUI and immediately display the selected event log file
s = system('Event_File_Reader_7_2.exe EVENTLOG.NLE'); % flat file format
s = system('Event_File_Reader_7_2.exe NEUR0003.DF1 NEUR0004.DF2'); % block file format with multiple files

% to read the data from the data file(s) and write directly to a CSV file without opening the GUI, using % commas as a delimiter
s = system('Event_File_Reader_7_2.exe EVENTLOG.NLE EventLog.CSV'); % flat file format
s = system('Event_File_Reader_7_2.exe NEUR0003.DF1 NEUR0004.DF2 EventLog.CSV'); % block file format with multiple files


% to read the data from the data file(s) and write directly to a CSV file without opening the GUI using tabs % as a delimiter
s = system('Event_File_Reader_7_2.exe EVENTLOG.NLE EventLog.CSV tab') % flat file format
s = system('Event_File_Reader_7_2.exe NEUR0003.DF1 NEUR0004.DF2 EventLog.CSV tab') % block file format with multiple files
```

4.3 Return values

The function returns an integer which is a value between 0 and 9.

0	Successfully wrote to file, ran the UI, or the user declined to overwrite a file.
1	Unauthorized file access
2	The requested file(s) could not be found
3	Invalid file collection. This may be due to entering multiple files of flat file format, multiple files in a collection having different 4 letter prefixes, discontinuous indices, an invalid extension (something other than ".DF1" or ".NLE"), etc.
4	Failed to write to CSV file
5	The settings file has been corrupted
6	No file has been loaded. Please load a file before requesting information about the event log file
7	The record you have requested is out of range. Please request a record greater than or equal to zero, and less than the number of records in the file.

5 Accessing event log files via DLL

There are 2 DLLs used to access data in an event log file: Event_File_Reader_7_2.dll (henceforth referred to as the main DLL) and Event_File_Reader_Dll_UM.dll which is a wrapper for the main DLL (henceforth referred to as the wrapper DLL). In some environments, the user will interact directly with the main DLL but in others the user will interact via the wrapper DLL.

5.1 Matlab

Users can directly interact with the main DLL from Matlab. The example is contained in the file EventFileReaderDllExampleScript.m.

Before running the script, the user should modify the following values in the User settings section:

dllFolder The full path of the folder in which the Event_File_Reader_7_2.dll is located

fileFolderName the name of the folder where your files are located.

recordToRetrieve the index of the record the user would like to retrieve from the file. This value is zero-indexed, so a value of zero will retrieve the first record. Therefore, if the user wishes to view the first record, they would assign this value to 0, for the second record, 1, etc.

In flat file format:

numberOfFiles Should always be 1, as there will only be 1 event log file per recording

The example script does the following:

1. Initializes the variables in set by the user
2. Loads the DLL
3. Load the file set in *fileName* variable
4. Check if loading in the file was successful. An error code of 0 indicates the file was loaded successfully. If the error code was non-zero, the script will print an error message.
5. Retrieve and print the header
6. Retrieve and print the number of records in the file
7. Retrieve a single record at the index of *recordToRetrieve* from the user settings section.
8. Display the retrieved record in a meaningful manner
9. Loop through all the records in the file and put them into an array of structs, where each struct represents a single record in the event log file.

Tested in version 2016a.

5.2 Python

Python users must access the main DLL via the wrapper DLL (Event_File_Reader_Dll_UM.dll). The user must place the main DLL (Event_File_Reader_7_2.dll) in the same folder as the python.exe executable. Type the following into the python shell:

```
>>> import sys
>>> print(sys.executable)
```

This will print the name of the folder where your python executable is running from, which is the location where you should place the Event_File_Reader_7_2.dll.

The wrapper dll can be placed in any folder.

In the section titled “User settings”, the user should set the following variables:

wrapperDll the name of the wrapper DLL (Event_File_Reader_Dll_UM.dll) including the full or relative path.

numberOfFiles The number of files you would like to view events from. In flat file format, this will always be 1.

fileNames **Flat file format*:*** the name of the event log file you wish to view (.NLE extension), including the or relative path. Replace "C:\\EVENTLOG.NLE" with your file name.

fileNameString **Block file format:** Replace "G:\\NEUR" The full or relative path of your files, including the 4 letter prefix.

recordToRetrieve the zero indexed value of the record you would like to retrieve e.g. to view the first record in the file, set this to zero.

Once the user has entered the above settings, the script is ready to be run. The script does the following:

1. Imports the necessary libraries
2. Initializes the variables that have been set by the user
3. Creates buffers that the DLL will use to store information
4. Loads the wrapper DLL
5. Loads the events from the files chosen by the user. If the files were loaded, it will print "Files was loaded successfully". Otherwise, it will print the error and exits the script.
6. Retrieves and prints the header from the event log file. If the call fails, it prints an error message and exits the script.
7. Retrieves and prints the number of records in the file. If the call fails, it prints an error message and exits the script.
8. Retrieves a single record from the file and prints each of the fields in the record. If the call fails, it prints an error message and exits the script.
9. Frees the DLL

This script can easily be extended to retrieve all the records from a file by calling the *GetIndexedRecord* function in a loop and storing them in an object.

Tested in Python version 3.6.5

5.3 C++

In Visual Studio 2017: C++ users must access the main event file reader DLL via the wrapper DLL. The user must place the main DLL (Event_File_Reader_7_2.dll) in the same folder as the C++ executable that is calling the DLL functions.

Open the Event_File_Viewer_7_2_Example.cpp class and copy it and the header (Event_File_Viewer_7_2_Example.h). Within the main function, you should modify the following values:

wrapperDll the name of the wrapper DLL (Event_File_Reader_Dll_UM.dll) including the full or relative path

maximumTimeToRunS If you wish to cancel this operation if it extends beyond a certain amount of time, set this to the number of seconds you wish it to cancel after. Set this to zero if you wish it to run to completion regardless of how long it takes.

recordToRetrieve	The index of the record you wish to print.

Flat file format:

numberOfFiles	This should be 1.
maxPathLength	Leave this as 260. Do not modify.
filesToLoad	Replace "C:\\EventLog.NLE" with the full or relative path of your event log file.

Block file format:

numberOfFiles	The number of files you wish to load
firstFile	The index of the first file in the collection you wish to load.
maxPathLength	Leave this as 260. Do not modify.
filename	Replace "G:\\NEUR" with the full or relative path of the files, including the 4 letter prefix.

The example script does the following:

1. Initializes the variables set by the user
2. Loads in the wrapper DLL
3. Initializes the size of pointer arrays that will be used by the DLL. This section should not be modified
4. Loads in the functions from the DLL
5. Checks that the DLL and each function were loaded in successfully
6. Loads the files
7. Checks if the file was loaded successfully. If not, it prints the error as a string.
8. Retrieves the number of records in a file
9. Retrieves and prints the header of the file (only in flat file format. Block format has no header)
10. Retrieves the record with the index specified by the user in *recordToRetrieve* and prints the information for the user.

This example can be easily extended to retrieve all the records in the file by calling the *GetIndexedRecord* function in a loop and storing each record in an object.

5.4 C#

In Visual Studio 2017: This program requires .NET Core 2.1 or higher. If you do not have this version, it can be downloaded here. Ensure that the target framework (Project->Properties->Application) is set to .NET Core 2.0.

You can copy the class Event_File_Reader_7_2_Main.cs into a new project. Add a reference to the main DLL (Event_File_Reader_7_2.dll) by right clicking the project name ->Add->Reference and selecting the DLL (you may need to browse to find it).

Inside the main function, modify the following variables:

Flat file format:

fileNames	Replace ”C:\\EventLog.NLE” with the full or relative path of your event log file

Block file format:

numberOfFiles	The number of files you wish to view
firstFileIndex	The index of the first file in the set you wish to view.
pathName	The path where the files you wish to view are stored
prefix	The 4 letter prefix of the file or group of files you wish to view

General:

cancelAfter	The maximum time limit in seconds to allow for loading of files. If the function takes longer than the maximum limit, it will be canceled.
recordToRetrieve	the zero indexed value of the record you would like to retrieve e.g. to view the first record in the file, set this to zero.

The example script does the following:

1. Initializes the variables set by the user
2. Initializes the data buffer where the data can be stored
3. Loads the files. If the function LoadFiles returns 0, the files have been successfully loaded. If the operation timed out, it will print “operation canceled” and exit the program. If an error occurred, the program will print the error and exit. If the operation completes successfully, it will continue to allow the user to retrieve the data.
4. Retrieves and prints the number of records in the file. If the function fails, the program prints the error message and exits the program.
5. Retrieves and prints the header. If the function fails, the program prints the error message and exits the program. Skip this call in block file format as there is no header.
6. Retrieves the chosen record set by the user in the *recordToRetrieve* variable. If this function fails, it prints an error message and exits the program.

This can easily be extended to retrieve all the records in a file by calling the *GetIndexedRecord* function in a loop and storing each record in an object or struct.

Appendix A

This appendix describes the different types of events that may appear in a file.

Error An error has occurred in the recording. Errors may be of the following types:
Dropped block type 0 – data transfer to the memory card driver was not fast enough so a single 64 kB block[1] of data was not written to the memory card.
Dropped block type 1 – the memory card was unable to accept data quickly enough so a 64 kB[1] block of data was dropped. If this happens more than once per 10GB of information, the user is probably using an unsuitable memory card.
Dropped block type 2 – a 64 kB[1] block of data was dropped because the logger needed to restart its data transfer hardware.
Motion sensor restart – the motion sensor chip needed to be restarted.

System Running Reserved

Stimulus In loggers that have the optional electrical stimulation module, this event occurs when an electrical stimulus is fired.

Fast reset This event occurs in loggers that support the option to rapidly reset the high pass filter if an input overload occurs.

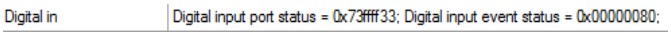
File started The timestamp of the start of a new 16 MB neural file.

Red-LED/ Green IR-LED event The LED on the logger was turned on or off if the user has elected to record such events.

Free text A custom event the user logged via the LoggerCommand3 program.

Digital in A digital input event has occurred. Such an event can be described in two different formats:

1. Hexadecimal representation of the digital input port and event status. The port status, when represented as a binary, describes the position of each of the 32 pins. The event status when represented as a binary describes which pins underwent events (for a given pin, 1 means an event has occurred and 0 means no event has occurred)



For example, in the above event, the input port status is 0x73ffff33 is represented in binary as 0111 0011 1111 1111 1111 1111 0011 0011 indicating that (zero indexed) pins 2, 3, 6, 7, 26, 27 and 31 are off and all

[1] The number of milliseconds per 64 kB block depends on the number of the channels in the system in the following table:

Number of channels	16	32	64
Number of ms	65.5	32	16

other pins are on. The input event status 0x00000080 is represented in binary as
0000 0000 0000 0000 0000 0000 1000 0000 meaning that there was an event only on pin 7 (zero indexed). Since the input port status of pin 7 is 0, this means that the event that occurred was a falling edge.

2. A string describing details of the events including the number of the pin on which it occurred, the nature of the event (rising or falling), and the name of the event.

Digital in	Digital in falling edge on pin number 8. Event name: Pin4Event;

This is the same event as the one above but displayed as a string. It says the event occurred on pin 8 because when displayed this way the pins are 1 indexed. Pin4Event is simply the name the user chose to give to an event on this pin.
The digital input information is always available in format 2, and is available in format 1 as well when the transceiver software has been set up for >4 digital inputs. The controls in the Display Control panel are used to set the display status of digital inputs as format 2, whereas the "Digital in" checkbox controls the display status of the event in format 1, if available.

Digital out A digital output pulse was sent from the transceiver. Details of the event (e.g. rising/falling) are recorded.

IRIG-J2 sequence An IRIG serial time sequence was sent to the digital output terminal of the transceiver.

Mode change The mode (e.g. recording, sleeping, monitoring) of operation of the logger was changed as detailed.

Clocks synchronized **Reserved.**

Recording parameters this record is sent at the start of each new recording session. It contains 17 parameters describing various aspects of the recording including firmware version, date, and channel map.

Parameter change Any parameter has been changed as detailed.

Automatic shutdown The logger has shut itself down.

GPS Timing Pulse In systems where the transceiver is equipped with a GPS clock[2], that clock can generate one pulse per second synchronized to GPS time and send each pulse as an event.

GPS Time String In systems where the transceiver is equipped with a GPS clock[2], the GPS can also send a time string instead of a pulse.

Startup Occurs when the LoggerCommand3 program initializes a system.

Warning	The message “data buffer is nearly full” appears when the write operation to the memory card has taken longer than usual. It is possible that data in a 64kB block are mistimed.
Information	The message “data buffer is nearly full” indicates that the write operation to the memory card has taken longer than usual but the data were correctly recorded.
PC-generated comment	A comment sent by the PC to the logger. This can contain information about the battery voltage or number of channels.
Tone generated	In instruments that have an audible sounder, a tone was activated. [2]

[2] Only some loggers contain this capability