**WISPR**

**Passive Acoustic Monitoring**

**File Formats**

Hardware Version: WISPR V3.0

Embedded Ocean Systems

Manual Version: 1.0

Aug 2024

Contents

[1. Configuration Files 3](#_Toc177118277)

[2. Data File Format 5](#_Toc177118278)

[2.1 File Header: 5](#_Toc177118279)

[2.2 Data Buffer Format 6](#_Toc177118280)

[2.3 Buffer Timestamp 6](#_Toc177118281)

[2.4 Matlab Script to Read Data Buffers 8](#_Toc177118282)

## Configuration Files

System configuration is stored on each SD card in a text file called **wispr.txt**. At startup the system reads the configuration file from SD1 to set the operating parameters. Operating parameters can be modified while the system is running using the command interface. For example, the ADC sampling rate can be changed using the ADC command. On controlled shutdown (using the EXI command) the system saves the current configuration to SD card to preserve the current settings for the next startup. If the system is powered down unexpectantly, the current configuration will be lost and the last saved configuration will be used on the next startup. The configuration file is saved on the current active SD card. For example, if SD1 is full the configuration will be saved on SD2.

**Config File Format:**

The first line of the ascii configuration file must contain the line ‘WISPR 3.0 configuration’, which indicates the version number and type of file. System variables follow as (variable: value) pairs on each line for the configuration file.

A typical configuration file looks like this:

WISPR 3.0 configuration

sensor\_id: PAM1

platform\_id: EOS1

start\_state: 2

start\_mode: 3

adc\_type: LTC2512

adc\_sample\_size: 3

adc\_sampling\_rate: 200000

adc\_channels: 1

adc\_gain: 0

adc\_decimation: 4

adc\_timestamp: 0

secs\_per\_file: 10

max\_file\_size: 102400

run\_time: 40

duty\_cycle: 60

debug\_level: 1

fft\_size: 512

fft\_overlap: 0

fft\_window\_type: 0

rtc\_aging: 0

rtc\_freq: 32767

msg\_baud: 9600

msg\_termination: 15

msg\_handshake: 1

Configuration variables are:

|  |  |  |
| --- | --- | --- |
| **Variable** | **Description** | **Value** |
| sensor\_id | User defined sensor ID | Max 16 chars |
| platform\_id | User defined platform ID | Max 16 chars |
| start\_mode | Mode when system starts | 0 to 255 |
| start\_state | State when system starts | 0 to 255 |
| adc\_type | ADC description | Max 16 chars |
| adc\_sample\_size | ADC data sample size in bytes | 2 or 3 |
| adc\_sampling\_rate | ADC sampling rate in Hz | 1000 to 300000 |
| adc\_gain | Preamp gain (0,6,12,18 dB) | 0, 1, 2 or 3 |
| adc\_decimation | ADC decimation factor | 4, 8, 16 |
| adc\_timestamp | ADC buffer timestamp size | 0 or 6 bytes |
| secs\_per\_file | Duration of data file in secs | 10 to 3600 |
| max\_file\_size | Max file size in 512 byte blocks | 2\*1024000 |
| run\_time | Data collection duration (secs) | >= 0 |
| duty\_cycle | DAQ duty cycle (secs) | >= run\_time |
| debug\_level | User defined debug print level | 0, 1, 2, or 3 |
| fft\_size | PSD FFT size | 128, 512, 1024 |
| fft\_overlap | PSD FFT Overlap | 0 to fft\_size |
| fft\_window\_type | PSD FFT Window type | 0, 1, or 2  (RECT,HANN,HAMMING) |
| rtc\_aging | RTC clock aging coefficient | -128 to 128 |
| rtc\_freq | RTC clock frequency in Hz | 32768 |
| msg\_baud | Command port baud rate | 9600 |
| msg\_termination | Command termination char | 10,13 or 15  (NL,CR,or CR\_NL) |
| msg\_handshake | Require message handshaking | 1 or 0 |

**Config File Values:**

All the numeric fields in the configuration file are decimal numbers. The fields named ***start\_state*** and ***mode*** are numbers comprised of bit fields. For example, a mode value of decimal 2 (0x02 in hexadecimal or 00000010 in binary) has the bit field 1 set, and all other bit fields unset. A state value of decimal 3 (0x03 in hexadecimal or 00000011 in binary) has the bit fields 0 and 1 set, and all other bit fields unset.

The bit fields definitions for the ***start\_state*** and ***start\_mode*** fields in the config file are as follows:

// Wispr State Flags

#define WISPR\_OFF\_STATE 0x00

#define WISPR\_RUN\_STATE 0x01

#define WISPR\_IDLE\_STATE 0x02

#define WISPR\_PAUSE\_STATE 0x04

#define WISPR\_SLEEP\_STATE 0x08

#define WISPR\_RESET\_STATE 0x10

// Wispr Mode Flags

#define WISPR\_DAQ\_MODE 0x01

#define WISPR\_LOG\_MODE 0x02

#define WISPR\_PSD\_MODE 0x04

#define WISPR\_DTX\_MODE 0x08

#define WISPR\_SLP\_MODE 0x10

#define WISPR\_GLIDER\_MODE 0x80

## Data File Format

ADC data is stored on the SD card in files of specified duration. Data files have a .dat file name extension. SD cards must be formatted as ExFAT.

The file name specifies the start date and time of the data in the file. For example, a file named *WISPR\_210218\_140500.dat* contains data starting at time 14:05 (Hour 14, minute 05, second 00) on Feb 21, 2021.

Each data file contains a 512 byte ascii file header followed a fixed number of data blocks. Data blocks are vectors of binary integers stored in IEEE little-endian format.

The data file consisting of N data buffers has format:

|  |  |  |
| --- | --- | --- |
| Description | Size in bytes | File offset in bytes |
| File Header | 512 | 0 |
| Data Buffer 1 | buffer\_size | 512 |
| Data Buffer 2 | buffer\_size | buffer\_size + 512 |
| Data Buffer 3 | buffer\_size | 2\*buffer\_size + 512 |
| … | buffer\_size | … |
| Data Buffer N | buffer\_size | (N-1)\*buffer size + 512 |

## File Header:

The ascii file header defines the parameters required to read and interpret the data file. A typical file header will look like this:

% WISPR 3.0

sensor\_id = 'EOS';

platform\_id = 'EOS';

second = 1725485740.138920;

file\_size = 70291;

buffer\_size = 23040;

samples\_per\_buffer = 7678;

sample\_size = 3;

sampling\_rate = 200000;

adc\_type = 'LTC2512';

adc\_vref = 5.000000;

adc\_df = 4;

gain = 0;

timestamp = 0;

The field 'time' is the data start time in seconds Linux epoch time. The fields ‘platform\_id’ and ‘sensor\_id’ are user defined fields taken from the configuration file.

The variable 'file\_size' is the number of 512 byte data blocks in the file. The total data file size is always a multiple of 512 bytes blocks.

Each data buffer is of length 'buffer\_size' bytes, which is always a multiple of 512 bytes blocks. The total number of data buffers in the file is:

N = file\_size\*512 / buffer\_size;

Each data buffer contains a fixed number of samples defined by the header variable ‘samples\_per\_buffer’. Each sample is of fixed size in bytes, defined by the variable ‘sample\_size’. The sample size can be 2 or 3, corresponding to 16 or 24 bit signed integers.

## Data Buffer Format

A data buffer with M = samples\_per\_buffer samples has format:

|  |  |  |
| --- | --- | --- |
| Description | Size in bytes | File offset in bytes |
| Data Sample 1 | sample\_size | 0 |
| Data Sample 2 | sample\_size | sample\_size |
| Data Sample 3 | sample\_size | 2\*sample\_size |
| … | sample\_size | … |
| Data Sample M | sample\_size | (M-1)\*sample\_size |
| Buffer Timestamp | timestamp | M\*sample\_size |

Samples are either 16 or 24 bit signed integers sampled by the ADC at the specified sampling frequency (sampling\_rate) in Hz. At the end of each sequence of samples is an optional buffer timestamp, as described in the next section.

## Buffer Timestamp

Each data buffer has an optional buffer timestamp, which specifies the time that the buffer finished and the start of the next buffer. There is no time gap between buffers. The data file header variable ‘timestamp’ defines the size in bytes of the timestamp value. If ‘timestamp = 0’, then there is no timestamp. If timestamp = 6, then a 6 bytes timestamp is appended to the end of each data buffer. If timestamp = 8, then an 8 bytes timestamp is appended to the end of each data buffer.

The 6 byte timestamp contains 2 leading bytes (a 16 bit integer) representing the seconds from the start of the data file and 4 bytes (a 32 bit integer) representing microsecond from the start of the second. The timestamp value is the time at the end of the data buffer, therefore the start time of the beginning of the next buffer.

buffer\_duration = samples\_per\_buffer / sampling\_rate;

seconds = fread(fp, 1, 'uint16' );

useconds = fread(fp, 1, 'uint32' );

buffer\_start\_time = t0 + (second + (0.000001 \* useconds)) – buffer\_duration;

where t0 is the epoch time for the start of the data file.

The 8 byte timestamp contains 4 leading bytes (a 32 bit unsigned integer) representing the epoch seconds and 4 bytes (a 32 bit unsigned integer) representing microsecond from the start of the second.

buffer\_duration = samples\_per\_buffer / sampling\_rate;

seconds = fread(fp, 1, 'uint32' );

useconds = fread(fp, 1, 'uint32' );

buffer\_start\_time = (second + (0.000001 \* useconds)) – buffer\_duration;

## Matlab Script to Read Data Buffers

function [hdr, data, time, stamp] = read\_wispr\_file(name, first, last)

%

% matlab script to read wispr data from a dat file

%

% A data file consists of an ascii file header followed by binary data

% buffers. The ascii header is formatted as matlab expressions.

% The binary data words are formatted as signed 16 or 24 bit integers.

%

% The data file format is:

% - 512 byte ascii header.

% - adc buffer 1

% - adc buffer 2

% ...

% - adc buffer N

% where N is the number of adc buffers per file

%

% The number of adc buffers is defined as

% number\_buffers = file\_size\*512 / buffer\_size;

%

% The total data file size is always a multiple of 512 bytes blocks.

% The variable 'file\_size' is the number of 512 blocks in the file.

%

% Each adc buffer is of length 'buffer\_size' bytes.

% The adc buffer is always a multiple of 512 bytes blocks

% (32 blocks in most cases).

% Each adc buffer contains a fixed number of sample (samples\_per\_buffer).

% Each sample is of fixed size in bytes (sample\_size).

% The sample size can be 2 or 3.

% If 3 byte samples are used, there will be extra bytes of padded

% at the end of each adc buffer.

% The number of bytes of padding is defined as:

% padding\_per\_buffer = buffer\_size – (samples\_per\_buffer \* sample\_size);

%

% File header info is returned in hdr

% Data and time are returned as a matrix of

% size [samples\_per\_buffer, (last - first)]

%

fp = fopen( name, 'r', 'ieee-le' );

% back compatability

instrument\_id = [];

location\_id = [];

sensor\_id = [];

platform\_id = [];

timestamp = 0;

time = [];

second = 0;

usec = 0;

adc\_type = 'LTC2512';

adc\_df = [];

decimation = [];

channels = 1;

% read and eval the ascii header lines

for n = 1:18

str = fgets(fp); % read full line

% read ascii lines until a null is found,

% so header buffer must be null terminated

if( str(1) == 0 )

break;

end

%fprintf('%s', str);

eval(str);

end

if(sample\_size == 2)

q = adc\_vref/32767.0; % 16 bit scaling to volts

fmt = 'int16';

elseif(sample\_size == 3)

q = adc\_vref/8388608.0; % 24 bit scaling to volts

fmt = 'bit24';

elseif(sample\_size == 4)

q = 1.0;

fmt = 'int32';

end

% The number of adc buffers in the file

number\_buffers = file\_size\*512 / buffer\_size;

% The number of bytes of padding after each adc buffer, if any

padding\_per\_buffer =

buffer\_size - timestamp - (samples\_per\_buffer \* sample\_size);

% buffer duration in secs

dt = 1.0 / sampling\_rate;

duration = samples\_per\_buffer \* dt;

% fill the header structure

hdr.time = time; % back compatability

hdr.second = second;

hdr.usec = usec;

hdr.sensor\_id = sensor\_id;

hdr.platform\_id = platform\_id;

hdr.instrument\_id = instrument\_id;

hdr.location\_id = location\_id;

hdr.file\_size = file\_size;

hdr.buffer\_size = buffer\_size;

hdr.samples\_per\_buffer = samples\_per\_buffer;

hdr.sample\_size = sample\_size;

hdr.sampling\_rate = sampling\_rate;

hdr.channels = channels;

hdr.gain = gain;

hdr.adc\_type = adc\_type;

hdr.adc\_vref = adc\_vref;

hdr.adc\_df = adc\_df;

hdr.decimation = decimation;

hdr.adc\_timestamp = timestamp;

hdr.number\_buffers = hdr.file\_size \* 512 / hdr.buffer\_size;

hdr.buffer\_duration = samples\_per\_buffer \* dt;

hdr.file\_duration = hdr.buffer\_duration \* hdr.number\_buffers;

data = [];

time = [];

stamp = [];

% check to make sure first and last are valid

if( first >= number\_buffers )

return;

end

% if first == 0 then just read the header

if(first <= 0)

return;

end

% read all buffers if last is zero

if(last == 0)

last = number\_buffers;

end

% past end of file

if( last > number\_buffers )

last = number\_buffers;

end

% seek to the start of data

% header is always 512 bytes

fseek(fp, 512 + (hdr.buffer\_size \* (first-1)), -1);

% start time

t0 = (second + usec \* 0.000001) + (first - 1)\*duration;

% number of buffer to read and concatenate

num\_bufs = last - first + 1;

for n = 1:num\_bufs

% read a data buffer

raw = fread(fp, samples\_per\_buffer, fmt ); % data block

if( length(raw) ~= samples\_per\_buffer )

break;

end

% read buffer timestamp, which is time the buffer finished,

% so remove the buffer duration

if( timestamp == 8 )

s = fread(fp, 1, 'uint32' ); % epoch second

us = fread(fp, 1, 'uint32' );

stamp(n) = (s + 0.000001 \* us) - duration;

elseif( timestamp == 6 )

s = fread(fp, 1, 'uint16' ); % secs from start of file

us = fread(fp, 1, 'uint32' );

stamp(n) = ((second + s) + 0.000001 \* us) - duration;

else

stamp(n) = t0 + (n-1) \* duration;

end

% read padding, if any

junk = fread(fp, padding\_per\_buffer, 'char');

% add raw data buffer as a column to data matrix

data(:,n) = double(raw)\*q;

% add a time column to the time matrix

time(:,n) = stamp(n) + dt\*(0:(samples\_per\_buffer-1));

end

fclose(fp);

return;