MATLAB SON Library

Malcolm Lidierth
King's College London
malcolm.lidierth@kcl.ac.uk
Updated July 2009 (version 2.32)

sigTOOL

This document explains how to use the SON library from the MATLAB command line and in user-written scripts. The library has also been incorporated into sigTOOL which provides GUI support, data visualization and an integrated environment for sharing open-source MATLAB functions for neurophysiological data analysis. It also provides for easy transfer of data from MATLAB to external programs such as Excel, SigmaPlot and Illustrator.
sigTOOL supports several file formats in input: CED Spike2 and Signal files, Axon Instruments pClamp files, Alpha Omega and Plexon Instruments formats as well as various multimedia formats such as WAV. sigTOOL is available from http://sigtool.sourceforge.net.
For a full description of sigTOOL see:

M. Lidierth (2009). sigTOOL: a MATLAB-based environment for sharing laboratory-developed software to analyze biological signals. *Journal of Neuroscience Methods* **178**, 188-196

NOTE: The MATLAB Central site no longer allows distribution of compiled code. If you use the SON32 library supplied from that site, you will need either to compile the files yourself or download sigTOOL which does include the compiled files. For further details see the section on the SON32 library below.

The most recent version of this software is available at:
http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=13932&objectType=FILE
Comments/bug reports can be posted on that site.

This pack contains two libraries.

SON2
SON2 provides a set of low-level routines that read from a SON file. These use standard MATLAB code, so they can be run under any operating system supported by MATLAB. The library will read both Windows and Mac Spike2-generated files. The new version includes better online help, improved error handling and a number of optional input arguments.

Changes in Version 2.2

In SON2 version 2.2, some functions have been improved for execution speed and memory use. This version introduces a 'mat' option that allows data from a SON file to be written to a standard MATLAB MAT-file and a SONImport function that imports all data from a file to a MAT-file.

Compatibility:
Sampling intervals are now returned in microseconds in the channel headers, not seconds as before. Timestamps from event, level and marker channels are returned as 32 bit signed integers if the 'Ticks' option is selected (in MATLAB version 7 or higher. Earlier versions still return double precision results)
Waveforms (ADC or RealWave and Marker) are returned as column vectors, not row vectors. With triggered sampling, each column (not row) corresponds to a sampled epoch. This facilitates memory mapping of a generated MAT-file (MATLAB saves data column-wise).
The functions have been updated to read SON filing systems up to Version 8 (June 2006) although only limited testing has been done with Version 7 & 8 SON files.

SON32
SON32 is a Windows only package. It allows MATLAB to read and write SON files (and to create them from scratch) by calling the functions in the son32.dll file that is supplied by CED with Spike2 for Windows. Son32.dll is the application extension that Spike2 uses to read/write files.

Comparison of the SON2 and SON32 libraries:

SON2	SON32
Use under any operating system (tested on Windows XP, Mac OS 10.4.8, Fedora 7 Linux).	Windows only
Reads SON files.	Reads/writes/creates SON files
Requires MATLAB version 6.5 or higher	Requires MATLAB version 7.1 or higher
Can be used to memory map data, saving space and speeding up MATLAB (Version 7 or higher only)	No memory mapping
Easy to use. Can be used with minimal knowledge of SON file structure	Designed for programmers – knowledge of the SON filing system is expected
Stand-alone, low-level code	Requires son32.dll which is supplied with Spike2 for Windows (Cambridge Electronic Design)
Can import data to a standard mat-file	

Both libraries include detailed online help.

USING THE SON2 LIBRARY

- **Opening a file**

The SON file should be opened using the MATLAB 'fopen' command e.g.

fid=fopen(*'demo.smr'*); reading only

If you are working on a non-Windows system you may also need to set the machine format in the call to fopen.

e.g.

fid=fopen(*'demo.smr','r','ieee-le'*)

to read a PC file on a Mac.

- **Importing a Spike file into a MATLAB MAT-file**

Simply call

SONImport(*fid*);

Or

SONImport(*fid, Options{:}*);

Where Options are the same as those specified for SONGetChannel below (but do not use the 'mat' option here).

All data from the Spike2 file will be written to a standard Version 6 MAT-file. The new file will have the same name and path as the Spike2 source file but will have the '.mat' file extension rather than '.smr'. Note SONImport will overwrite the mat-file if it already exists.

When data has been imported to a MAT-file you can use the standard MATLAB load command to access the data and its header e.g.

load myfile chan1;
load myfile head1;

The mat-file may also be memory mapped using memmapfile (v7 or higher). sigTOOL does all this for you and should be used in preference (see above).

- **Reading a channel**

Data is read using the SONGetChannel command. A call to SONGetChannel takes the form:

data=SONGetChannel(*fid*,*chan*);

or

[*data, header*]=SONGetChannel(*fid*,*chan*);

Where *data* will be returned with the data stored on the channel, *fid* is the file handle from fopen() and *chan* is the channel number - numbered form 1 to (maxchan-1) as in Spike2. *header* is an optional output argument and provides details of the channel type, sample rates etc.

SONGetChannel identifies the data type in the channel and calls one of a set of routines depending on the data type. These routines can also be called directly but this is discouraged – some common code is implemented in SONGetChannel rather than being included in all the other routines.

The SONGetChannel command also accepts optional arguments. These:

- Allow parts of the channel data to be loaded instead of all data
- Allow you to choose the time units for returning timestamps
- Let you choose whether to scale ADC data
- Allow a progress bar to be displayed when reading a file

- Reading parts of the data.

Within a SON file, data is stored in arbitrarily sized blocks. For continuously sampled data, you can read one, or a subset of the blocks as follows

data=SONGetChannel(*fid, chan, 10*);

reads block 10 only

data=SONGetChannel(*fid, chan, 10, 20*);

reads blocks 10 to 20 inclusive.

If sampling on the relevant channel was done in triggered-mode, these calls would return epoch 10 or epochs 10-20 inclusive (regardless of the disc blocks involved – the SON library sorts those out automatically).

- Setting the time units

By default, the library returns times in seconds but you can choose microseconds, milliseconds, seconds or clock ticks e.g.:

data=SONGetChannel(*fid, chan, 'microseconds'*);

returns all data on the channel scaled in µs.

data=SONGetChannel(*fid, chan, 10, 'ticks'*);

returns block 10 only scaled in clock ticks

[data, header]=SONGetChannel(*fid, chan, 10, 20, 'seconds'*);

returns blocks 10-20 inclusive scaled in seconds.

Changes in Version 2.2

The timeunits affect the *data* and the values returned in *header* except for the *header.sampleinterval* field which, from Version 2.2 onwards, is always returned in microseconds. If you want the sample interval in seconds for compatibility with previous versions of this library, uncomment the last line of SONGetSampleInterval.m to revert to the old standard.

- Scaling data

For ADC data, the SON file stores a scale and offset to convert 16-bit integer values to floating point real world values. The scale option causes data to be returned appropriately scaled e.g. :

data=SONGetChannel(*fid, chan, 'scale'*);

[data, header]=SONGetChannel(*fid, chan, 10, 20, 'seconds','scale'*);

• Progess bar

Selecting progress displays a progress bar during a read operation. If you do not specify the number of blocks (or epochs) to read, the progress bar will only appear if the read request involves loading more than 10 blocks.

[data, header]=SONGetChannel(*fid, chan, 1, 200, 'seconds','scale','progress'*);

You can put the optional arguments into a cell array e.g.:

Options{1}='scale';
Options{2}='milliseconds';
Data1=SONGetChannel(*fid, 1, Options{:}*);
Data2=SONGetChannel(*fid, chan, 1, 100, Options{:}*);

- MAT-file creation

New in Version 2.2

The 'mat' option should be followed by the name of the MAT-file to which data will be appended. Data are written in Version 6 format – without data compression. For waveform channels, data are written epoch by epoch.

SONGetChannel(fid,2,'progress','mat','myfile.mat');

Data will be written to a variable in the MAT-file called chanx, in this case chan2. (Note: if output arguments are present, data will be returned empty for waveform channels)

When data has been imported to a MAT-file you can use the standard MATLAB load command to access the data e.g.

load myfile chan4;

The header is not saved. User SONImport in preference to this option.

Reading single channels with SONGetChannel

For convenience, the examples below do not use optional input arguments.

ADC data

data is returned either as a single column vector for continuously sampled data or as a matrix of column vectors[1] for triggered sampling i.e.,

Changed in Version 2.2

Sample1
Sample2
Sample3
Sample4
Sample5 etc

or

Epoch 1 Sample 1	Epoch 2 Sample 1	Epoch 3 Sample 1	……..	……..
Epoch 1 Sample 2	Epoch 2 Sample 2	……..	……	……
Epoch 1 Sample 3	……	……..	……..	……..etc

When the source file is derived from frame based acquisition software (such as CED's Signal), the Epochs in each column of *data* should be aligned with the trigger. However, this will not generally be the case with Spike2 derived files. The triggered sampling mode introduced in Spike 2 Version 5 produces sweeps of variable length and the trigger will

occur at variable distances into each column of *data* (the sweep time and pre-trigger time set in Spike2 are treated as minimum values for each sweep – not constant values from sweep to sweep).

RealWave data

RealWave data is read and stored in 32 bit floating point (single precision) format. The inputs and outputs are as for ADC data (above) except that *data* is single precision floating point and the *header* does not contain a *scale* or *offset* field. Instead a *min* and *max* field are present that give expected minimum and maximum values in *data* (Note: RealWave channels written in Spike2 may contain *scale* and *offset* of the source channel in *max* and *min.*). For RealWave data *header.kind* is set to 9.

Event data

Event data is read as 32-bit integers giving the time of event in clock ticks returned in 64-bit floating point (double precision) format with the times given in seconds (by default) or in the time units selected by an optional argument to SONGetChannel (see above). Note that, if the selected time units are clock 'ticks', the time stamps will be returned as 32 bit signed integers in version 7 or higher of MATLAB rather than as floating point.

Change in Version 2.2

[*data {,header}*]=SONGetChannel(*fid, chan*);

where *data* is a column vector containing the times of each event in sequence expressed in seconds since the start of sampling.

If *header.kind*=4 data is of EventBoth type. The header will then contain a field called *h.initLow*. This defines the initial state of the channel: 1=low, 0=high.
If *initLow*=1 the first event will be a low to high transition. *data* will contain the times of low-to-high transitions in its odd numbered elements and high-to-low transitions in its even numbered elements.
If *initLow*=0 the first event will be a high to low transition. *data* will contain the times of high-to-low transitions in its odd numbered elements and low-to-high transitions in its even numbered elements.
The *h.nextLow* field is a flag used during sampling. Although copied to the header it is not used here.

Marker data

Marker data is treated in the same way as event data except that the returned *data* is a MATLAB structure containing two fields.

[*data {,header}*]=SONGetChannel(*fid, chan*);

returns a *data* structure containing for example

timings: [1x38 double or int32]

markers: [38x4 uint8]

with a channel with 38 markers.

The *data.timings* field is a column vector containing the marker times just as with event data. In the SON system, each marker event has associated with it four single byte markers. Each marker makes up a column vector in *data.markers* .
To access data for individual events use subscripting e.g.
data.timings(10) gives the time of the tenth event marker and *data.marker(10,3)* gives the value of marker 3 for event 10. You can access all markers in one go using range subscripts e.g. for channel 31 of the demo.smr file supplied with Spike2:

```
>> [d,h]=SONGetChannel(fid,31);
>> data.markers(:,1)'

ans =

thisisanexampleofdatasampling.Goodbye!

>>
```

ADCMark (WaveMark) data
ADCMark data is treated in the same way as marker data except that the returned *data* is a MATLAB structure containing three fields.

[*data {,header}*]=SONGetChannel(*fid, chan*);

returns a *data* structure containing for example

```
timings: [279x1 double or int32]          % 279 events
markers: [279x4 uint8]                    % each with four markers…
adc: [32x279 int16]                % …and 32 adc values
```

The *data.timing* and *data.marker* fields are the same as with ordinary marker data. The *data.adc* field contains the ADC data associated with each event. If the 'scale' optional input argument is given, ADC data will be scaled and offset and cast to double precision floating point.
The ADC data is organized as a series of column vectors, with one vector for each marker event. Access data for individual marker events as before. For the ADC data you can access individual values.

RealMark data
RealMark data is treated in the same way as ADCMark data but returns real valued data in *data.adc*.

[*data {,header}*]=SONGetChannel(*fid, chan*);

returns a *data* structure containing for example

timings: [13481x1 double or int32]	% 13482 events
markers: [13481x4 uint8]	% each with four markers…
real: [1x13481 single]	% …and one single precision value in this case

TextMark data
Text mark data allows a string of characters to be attached to a marker event.

[*data {,header}*]=SONGetChannel(*fid, chan*);

returns a *data* structure containing e.g.:
timings: [41x1 double or int32]
markers: [41x4 uint8]
text: [80x41 uint8]

The length of the strings in *data.text* is variable and determined when reading the SON file.

CONVERTING DATA

Wave data
Routines are included that convert between 16-bit integer, and single and double precision floating point representations.

- **Coverting to floating point.**

Scaling will be done automatically, and more efficiently, if the 'scale' option is given to SONGetChannel. However, if integer data has been returned

[*dataout {,headerout}*]=SONADCToSingle(*datain {,headerin}*);
and
[*dataout {,headerout}*]=SONADCToDouble(*datain {,headerin}*);

convert 16-bit integer data to single and double precision floating point respectively. If present, the values of *headerin.scale* and *headerin.offset* are used scale *dataout* into units of *header.units.*
headerin is copied to *headerout* and values of the minimum and maximum values in *dataout* are added as fields to *headerout* as with RealWave data in the SON system. Output is assigned a channel type of 9 corresponding to RealWave data in SON regardless of whether *dataout* is single or double precision.
(Note: if *scale* is 1 and *offset* is 0 there is no advantage to these routines over the MATLAB single or double conversions).

- **Converting to 16-bit integer**

Converting from floating point to 16-bit integer is more difficult. Data needs to be scaled and an offset applied to make maximum use of the 16 bits available. This is done with:

[*dataout {,headerout}*]=SONRealToADC(*datain {,headerin}*;)

datain can be single or double precision. Regression is used to find values of *headerout.scale* and *headerout.offset* and *dataout* is a newly scaled version of *datain.* *dataout* is rounded to the nearest integer and always fills the range -32768 to 32767. Expect a quantization error of about (10*header.scale)/(2*65536) i.e ±1/2 LSB.
dataout and *headerout* can be written as a new ADC channel using SONCreateChannel. Scale and offset will be converted to single precision on disk.

- **Event and marker data**

To convert between event or marker times in clock ticks and seconds.
dataout=SONTicksToSeconds(*fid, datain, option)*;

option is a string or cell array of strings. Valid options are ticks (does nothing), microseconds, milliseconds or seconds (default).

MISCELLANEOUS

The following functions are used internally by the library:

***F*=SONFileHeader(*fid*);**
Returns a MATLAB structure containing the SON file header. See CED SON documentation for details.

***L*=SONChanList(*fid*)**
Returns a MATLAB structure array with one element for each active channel in the SON file. Each element is a structure giving some details about the channel:

```
number: 1                % the channel number as it would appear in Spike2
kind: 1                  % channel type
title: 'Sinewave'
comment: 'sinewave'
phyChan: 0               % physical port if appropriate
```

***I*=SONGetChannelInfo(*fid,chan*);**
Returns the disk SON channel header as a MATLAB structure. See CED SON documentation for details.

***B*=SONGetBlockHeaders(*fid,chan*)**
Returns the disk block headers as a MATLAB matrix. See CED SON documentation for details but note that *B* will contain a pointer to the start of each block not to the previous and succeeding block e.g.

```
B =
5120        % disk pointer to start of block
100         % start time (clock ticks)
2549100     % end time (clock ticks)
1           % channel number
2550         % number of data points
```

SONGetSampleInterval and SONGetSampleTicks
[*interval {,start}*]=SONGetSampleInterval(*fid, chan*)
or
[*interval {,start}*]=SONGetSampleTicks(*fid,chan)*
return the sample interval and the time of the first sample in microseconds* and clock ticks respectively for channel *chan* in file *fid*.

Change in Version 2.2

SONTest(*'filename'*) or SONTest(*fid*)
Runs the CED SONFix.exe test utility on the file. If filename is e.g. 'c;\data*.smr' all files in the directory are tested. Files that are open for writing (in MATLAB or e.g. Spike2) will not be tested. (Note: always leave SONFix using its Exit button).

* Changed in version 2.2. Previously seconds

SONVersion
Displays a box with the version number of this SON library and the SON versions it supports

USING THE SON32 LIBRARY

To use this library you need a copy of son32.dll (supplied as part of Spike2 for Windows). This file has to be in the c:\Spike5 folder.

NOTE: The MATLAB Central site no longer allows distribution of compiled code. If you use the SON32 library supplied from that site, you will need either to compile the files yourself or download sigTOOL which does include the compiled files.
To compile the files set up a C compiler using mex –setup at the MATLAB command line. Then run the compile.m MATLAB file which invoke the compiler for each file.
Alternatively, download sigTOOL. The entire SON library will be found in the …sigTOOL\sigTOOL Neuroscience Toolkit\File\menu_Import\group_NeuroScience File Formats\son\ folder. Compiled files will be found in the …\SON32 subfolder.

The 'Additional Documentation' included with Spike2 describes the functions contained in son32.dll.

To access these functions using the MATLAB SON32 library

1. Load the library by running SONLoad
2. Open the file using SONOpenOldFile or SONCreateFile (not fopen)

Most of the functions in the dll are called using the MATLAB built-in calllib. If you use calllib directly, you must pre-allocate the output arrays. If this is not done properly, segmentation errors will occur and MATLAB will crash and/or become unstable. The supplied m-files take care of memory allocation for you.

Some functions can not be called with calllib as it can be used only with scalar structures. Gateway functions have been written in C to call these functions (the compiled files have the MATBAL version 7.1 compatible mex32 file extension. Also included are compiled dlls for use with earlier MATLAB versions).

The main difference between the MATLAB calls and those made directly from C is that markers are returned as a structure of arrays instead of as an array of structures. This allows the arrays to be used as arguments to the MATLAB built-in functions more easily. The online help provides further details.

To unload the library (and free memory) call unloadlibrary('son32').

Pitfalls:
Remember that SONOpenOldFile (and SONCreateFile) return handles that are internal to son32.dll. They are not valid MATLAB file handles.
Channels are numbered from 0 to maxchan-1 in the SON32 library (not 1 to maxchan as with the SON2 library).