MATLAB SON Library

Malcolm Lidierth
King's College London
Updated January 2003

This library contains routines written in MATLAB (version 6.1 release 12.1) that read data from a CED SON file (up to version 6). Limited support is also given to write data from MATLAB to an existing SON file.
The routines have been used on a PC running Windows 98 with SON files from versions 3 through 6 but no guarantees are given. Use the library at your own risk.
CFS files from CED's Signal or Sigavg software can be read if converted to SON format in Spike2 or using CED's C2S.EXE program.

- **Opening a file**

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

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

or

fid=fopen(*'demo.smr','rb+'*); reading and writing

All the functions to deal with SON files are provided as MATLAB m-files with the prefix SON. Most functions requires you to supply a value for *fid* from the fopen command above.

- **Reading a channel**

Data is read by calling SONGetChannel e.g.

data=SONGetChannel(*fid, channelnumber*);

Here, channelnumber is the same as the channel number assigned in the CED Spike time-view for the file. Alternatively, call:

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

In this case a data header is also created and returned. Its structure depends on the type of data read.

- **Exporting data**

At present only waveform data can be written to a SON file and these need to be based on an existing template channel in the file.

newchannel=SONCreateChannel(*fid, templatechannel, data {,header}*);

Here, *templatechannel* will usually be the channel number of the original source data. Data contained in *data* is written to a new channel by appending it to the end of the file. 16-bit integer (ADC) data and 32-bit floating-point (RealWave) data can be written. The optional *header* input is described below.

- **Manipulating files and data**

A number of support routines are provided e.g. to convert clock ticks to seconds and *vice versa*. These are described below.

Waveform data is returned by the SON routines in the most memory saving format e.g. as 16-bit integer for ADC data. For the most part, MATLAB expects double precision floating point input and can not handle this data until it is converted to that

format. Routines are included to convert between data types using the scaling factors stored in the SON source file.

READING DATA

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

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

where {*header*} is optional.

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 (see below).

ADC data
ADC data is read and stored in signed 16-bit integer format (int16). Data is read directly by a call to SONGetADCChannel.
data is returned either as a single row vector for continuously sampled data or as a matrix of row vectors for triggered sampling i.e.,
either

Sample 1	Sample 2	Sample 3	Sample 4	Sample 5 etc..

or

Frame 1 Sample 1	Frame 1 Sample 2	Frame 1 Sample 3	……..	……..
Frame 2 Sample 1	Frame 2 Sample 2	……..	……	…….
Frame 3 Sample 1	……	……..	……..	……..etc

header is a MATLAB structure. Its contents depend on the channel type and, in some cases, the version of the SON filing system being used.

```
FileName: 'd:\temp\example.smr'   % the name of the path and file this data was read
                                  % from
system: 'SON3'                    % the filing system, in this case SON version 3
FileChannel: 1                    % the channel number - as in Spike2
phyChan: 0                        % the physical channel - in this case ADC input #0
kind: 1                           % the channel type - 1 for ADC data
comment: 'Sinewave'               % the channel comment supplied from e.g. Spike2
title: 'Sinewave'                 % channel title (as above)
sampleinterval: 0.0100            % the interval between samples in seconds
scale: 1                          % scale….
offset: 0                         % …offset…..
units: 'Volts'                    % …and units y(units)=(data*scale/6553.6)+offset
start: 0.0010                     % the times (in seconds) of the first….
stop: 25.4910                     % …. and last sample in each row (frame) of data.
                                  % N.B. Only one frame in this case
```

transpose: 0 % 0=*data* is row vector based. 1=*data* is column
% vector based.

Note that the *start* and *stop* fields are row vectors with one entry for each row of *data* i.e. for each frame.

The *transpose* flag is intended to keep track of the orientation of waveform *data*. The row vector organization is best for some functions while column vectors are better for others. With large data arrays, frequently transposing them will waste too much time. It may be better to transpose the data in the main workspace and set the transpose flag so that you know the orientation of the data. (Note: *transpose* is defined in SONGetChannel. A direct call to SONGetADCChannel, or similar channel-type specific functions, will leave *transpose* undefined).

header is a MATLAB structure data type. You can therefore easily add your own fields to it e.g.
header.myfield='add a new string field';

You can get more information about each channel and its structure on disk by calling the SONChannelInfo and SONGetBlockHeaders functions, which are described later.

Integer data can be converted to floating point simply by using double(*data*) or single(*data*) but this leaves the values unaffected. The SONADCToDouble and SONADCToSingle functions do the conversion and scale the output according to the values of scale and offset stored in the SON file. Both functions take the same form e.g

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

headerin is a structure such a the header returned by SONGetChannel. Only the scale, offset and kind fields need be present.

In both cases the output is
(datain*headerin.scale/6553.6)+headerin.offset

If present *headerin* is copied to *headerout*. The following fields are added or changed:
headerout.max % maximum value in *dataout*
header.out.min % minimum value in *dataout*
headerout.kind % Set to 9 (for both double and single precision)

The output from SONADCToSingle can be written to a SON file as a new RealWave channel using SONCreateChannel (see below).

RealWave data

RealWave data is read and stored in 32 bit floating point (single precision) format. Data can be read through SONGetChannel or directly by a call to SONGetRealWaveChannel. The inputs and outputs are as for SONGetADCChannel (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 and returned in 64-bit floating point (double precision) format with the times given in seconds. Data can be read through SONGetChannel or directly by a call to SONGetEventChannel.

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

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

The optional *header* contains the following fields
FileName: 'demo.smr'
system: 'SON3'
FileChannel: 2
phyChan: 0
kind: 2,3 or 4
comment: 'Stimulus pulse, once per sinewave cycle'
title: 'Stimulus'
transpose: 0

Version 1.01 and above

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.

You can convert between seconds and clock ticks using the SONSecondsToTicks or SONTicksToSeconds functions described below.

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. Use SONGetChannel or a direct call to SONGetMarkerChannel.

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

returns a *data* structure containing for example

timings: [1x38 double]
markers: [4x38 char]

with a channel with 38 markers.

The *data.timings* field is a column vector containing the marker times in seconds 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(id,31);
>> data.markers(:,1)'

ans =

thisisanexampleofdatasampling.Goodbye!

>>
```

The header fields for marker data are:
FileName: 'demo.smr'
system: 'SON3'
FileChannel: 31
phyChan: 16
kind: 5
comment: 'Keyboard'
title: 'Keyboard'
transpose: 0

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. Use SONGetChannel or call SONGetADCMarkerChannel directly.

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

returns a *data* structure containing for example

```
timings: [279x1 double]      % 279 events
markers: [279x4 char]        % each with four markers…
adc: [279x32 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. The ADC data is organized as a series of row 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 e.g

```
>> data.adc(5,15)
ans =
   -65
```

gives the value for the 15th adc sample of the 5th marker event.

```
>> plot(d.adc(5,:))
```

plots all the data associated with the 5th event.

```
>> plot(d.adc(5:10,:)')
```

plots all the data associated with the 5th through 10th event.

Markers are generally numbers rather than characters. Use int8 or uint8 to inspect them e.g.

```
>> int8(d.markers)

ans =

    1    0    0    0
    1    0    0    0
    1    0    0    0
    1    0    0    0
    1    0    0    0
--more--
```

The header for ADCMark data contains the following fields:

FileName: 'spike3.smr'
system: 'SON4'
FileChannel: 4
phyChan: -1
kind: 6
values: 32
preTrig: 10
comment: 'No comment'
title: 'untitled'
sampleinterval: 2.4000e-005
scale: 500
offset: 0
units: 'µV'
transpose: 0

These are the same as with ADC data but with the addition of:
header.values indicates the number of ADC samples for each event.
header.preTrig number of pre-trigger samples
Note the timing of the event in *data.timings* is for the first sample in *data.adc* and takes no account of the value of *header.preTrig.*

RealMark data
RealMark data is treated in the same way as ADCMark data but returns real valued data in *data.adc*. Use SONGetChannel or call SONGetRealMarkerChannel directly.

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

returns a *data* structure containing for example

```
timings: [13481x1 double]      % 13482 events
markers: [13481x4 char]        % each with four markers…
real: [13481x1 single]         % …and one single precision value in this case
```

header contains:

```
FileName: 'donotdelete.smr'
system: 'SON5'
FileChannel: 1
phyChan: 2
kind: 7
values: 1
preTrig: 0
comment: 'Organ bath measurement channel'
title: 'Tension'
sampleinterval: 0              % Note zero in this test file
min: 0
max: 3
units: 'g'
transpose: 0
```

TextMark data
Text mark data allows a string of characters to be attached to a marker event. Read text mark data through SONGetChannel or directly by SONGetTextMarkerChannel.

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

returns a *data* structure containing e.g.:
```
timings: [41x1 double]
markers: [41x4 char]
text: [41x80 char]
```

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

header contains e.g.:

FileName: 'donotdelete.smr'
system: 'SON5'
FileChannel: 32
phyChan: 0
kind: 8
values: 80
preTrig: 0
comment: 'User entered comments'
title: 'Comment'
transpose: 0

WRITING DATA

Waveform (ADC) and RealWave data can be written to an existing file if the source data array is structured in the same way as an existing channel. The existing channel serves as a template channel giving information about sample rate, internal disk block structure etc. As data is written to the file it is possible for the file to be corrupted. It is advisable to make sure you have a back-up copy of any file before attempting to write to it.

Data is written using:

newchannel=SONCreateChannel(*fid, SrcChan, data {, data.header}*);

Output
newchannel is the channel number of the new channel. There must be space in the SON file for a new channel. By default, SON channels have space for a minimum of 32 channels.

Input
fid is the file identifier from fopen. The file must be open for appending of new data (permission='rb+' in fopen).

SrcChan is the existing channel to use as a template. SrcChan can be either an ADC or RealWave channel.

data is the data array to write and should be in the format as that returned by SONGetChannel for SrcChan i.e have the same row and column structure (although it may be transposed, see below). If *data* is a 16-bit integer matrix an ADC channel is written. If *data* is single precision floating point, a RealWave channel is written. If *data* is

double precision it is rounded to single precision before being written. SON files must be version 6 for RealWave channels to be written.

data.header is optional. The channel header from *SrcChan* is always copied to *newchannel*. The channel title is altered by pre-pending 'MATLAB:' to it. The channel type is then updated to indicate ADC or RealWave data as appropriate.
Then, if *dataheader* is present:

1. For ADC channels, if present the values for *dataheader.scale* and *dataheader.offset* are written to disk.
2. For RealWave channels the values of *dataheader.min* and *dataheader.max* are written.

If *dataheader* is not present the values from *SrcChan* will be left unaltered in *newchannel*.

If *dataheader* is present, *dataheader.transpose* indicates whether data is row (0) or column (1) vector based. By default, a row vector organization is assumed as returned by the data reading functions described above.

Note that data is always appended to the existing disk file even if it contains deleted blocks. Blocks from deleted channels are not re-used. SON files can often be reduced in size by using the File Export As feature in Spike2.

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.**

[*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**

Two routines convert between event or marker times in seconds and clock ticks. These are:
dataout=SONTicksToSeconds(*fid, datain)*;
and
dataout=SONSecondsToTicks(*fid, datain*);

MISCELLANEOUS

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 seconds and clock ticks respectively for channel *chan* in file *fid*.

SONUpgradeToVersion6(*fid*);
Does a limited conversion of an existing SON file to version 6. This allows RealWave data to be written to it. You should make a back-up file before attempting the upgrade. Note that only ADC and ADCMark data are upgraded. Date and time fields in the file header are not upgraded. If the *creator* field in the file header contains '00000000' it is replaced with 'MAT-UP-6' but left untouched otherwise.

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).

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

ERROR HANDLING

In general, when the functions encounter an error they issue a warning and execute a return without assigning output variables a value. Program execution is not halted so you may get a series of warnings. The first is the one that is relevant.

The warning may be terminated by a MATLAB generated error. In general this will be because data has been requested from a non-existence file, channel etc. Control then returns to the MATLAB command line.

Revisions

1.01 21st January 2003
SONGetEventChannel handles event channels of type 4 (EventBoth data) correctly