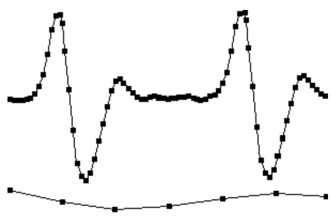
Data channel types

A SON file contains from 32 to 256 channels of data (not all channels need be used). The number of channels cannot be changed once a file has been created. The standard and backwards compatible number of channels is 32. The first channel is number 0. There are currently nine different types of data channel:

- 16-bit integer waveform data (called `Adc` throughout the SON library)
- Event data, times taken on the low going edge of a pulse (`EventFall`)
- Event data, times taken on a high going edge of a pulse (`EventRise`)
- Event data, times taken on both edges of a pulse (`EventBoth`)
- Markers, an event time plus four identifying bytes (`Marker`)
- 16-bit integer waveform transient shapes (we call this `AdcMark` data), an array of waveform data attached to a marker. In version 6 the waveform data may be multiple, interleaved channels.
- Real markers, an array of real numbers attached to a marker (`RealMark`)
- Text markers, a string of text attached to a marker (`TextMark`)
- 32-bit floating point waveforms (`RealWave`). These are new at version 6.

Waveform data



The waveforms you record are continuously changing voltages. The SON file format stores waveforms as a list of 16-bit signed integers (`Adc`) or 32-bit floating point numbers (`RealWave`) that represent the waveform amplitude at equally spaced time intervals. We also store a scale factor and offset for each channel to convert between integers and user defined units, the value of the data in user units is given by:

```
real value = integer * scale / 6553.6  +  offset
```

This scale factor was so that on systems where the 16-bit integer range corresponded to ±5 Volts, a scale factor of 1.0 and an offset of 0.0 produced a real value in Volts. The `scale` and `offset` allow us to read `RealWave` data as `Adc` and `Adc` data as `RealWave`.

The process of converting a waveform into a number at a particular time is called sampling. The time between two samples is the sample interval and the reciprocal of this is the sample rate, which is the number of samples per second. The dots in the diagram represent samples, the lines show the original waveform.

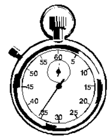
The sample rate for a waveform must be high enough to correctly represent the data. It can be demonstrated mathematically that you must sample at a rate at least double the highest frequency contained in the data. On the other hand, you want to sample at the lowest frequency possible, otherwise your disk system will very soon be filled. Unlike many data storage systems, the SON library allows you to save different waveform channels at different rates.

In the SON data model, waveform data is sampled at an integer multiple of the file clock tick. Each waveform channel can be sampled at a different multiple of this clock tick, thus all the waveform channels can be sampled at different rates.

Before version 6, all waveform channels were presumed to be sampled at a multiple of a time interval that was itself a multiple of the file clock tick period. The time interval was given by `usPerTime * timePerADC` *base time units. This modelled the situation where data came from an ADC (Analogue to Digital Converter) that ticked every* `timePerADC` *clock ticks. We define* `timePerADC` *later.*

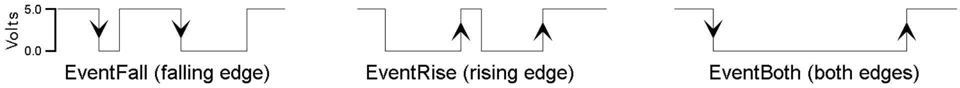
Each block of waveform data holds a start time, so waveform data need not be continuous. Two waveform blocks on the same waveform channel hold continuous data if the time interval between last sample in the first block and the first sample of the second block is equal to the sample interval.

Event data

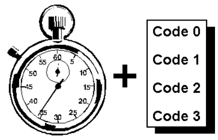


There are three types of Events that are stored in the same way: `EventFall`, `EventRise` and `EventBoth`. We refer to all three types as `Event` data. Events are 32-bit time stamps. If the value of the time stamp is `n`, this means a time of `n` clock ticks which is `n * usPerTime` base time units. Events can be either discrete points having a time of occurrence but no duration or they can mark the change of state of a signal between two levels. Generally the `EventFall` and `EventRise` forms of events are the more useful.

Event channel types



Marker data

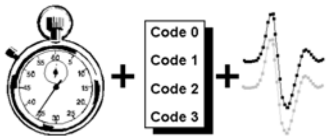


Marker events (`Marker`) are stored as 32-bit times, like events, but they have 4 additional bytes of information associated with each time. These additional bytes are used to hold information about the type of each marker. Markers typically hold user key press information, or the state of variables. For example, the CED VS software uses markers to hold the start time of stimulus presentations, and to hold the state of the independent variables that define the presentation. The Spike2 software uses the first byte as the ASCII code of a key press or data from the digital inputs.

Markers can be filtered by a user-supplied mask so that only those markers which meet a given set of criteria will be considered.

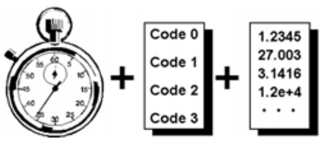
The SON library functions can read a `Marker` channel as an `Event` channel or as a `Marker` channel.

AdcMark data



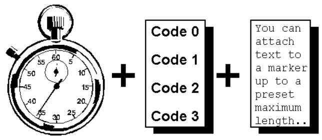
This type is a combination of waveform and marker data. It is stored as a 32-bit time and 4 bytes of marker information, followed by waveform points. From version 6 onwards, the waveform data may be from 1 up to 8 channels, stored as interleaved data. The waveform(s) typically holds a transient shape, and the marker bytes hold any classification codes required for the transient. The first point in the waveform data is sampled at the marker time. The SON library functions can use an `AdcMark` channel as if it were a waveform (only if there is a single channel of data), `Event`, `Marker` or `AdcMark` channel. To avoid alignment problems, we recommend that the number of points times the number of interleaved channels is an even number.

RealMark data



This is effectively `Marker` data to which is attached an array of real numbers. It is stored as a 32-bit time and 4 bytes of marker information, followed by an array of 4 byte real numbers. The SON library functions can use a `RealMark` channel as if it were an `Event`, `Marker` or `RealMark` channel. A `RealMark` channel with a single real value can be treated as a non equal-spaced waveform channel.

TextMark data



This is `Marker` data to which is attached a character string. It is stored as a 32-bit time and 4 bytes of marker information, followed by an array of characters. The character array is of fixed size, the strings stored in the arrays are terminated by a zero byte and can be of any length up to the array size-1. We recommend that you use array sizes that are multiples of 4 to avoid alignment problems. The SON library functions can use a `TextMark` channel as if it were an `Event`, `Marker` or `TextMark` channel.

Reading data

Data is read back from a SON file by specifying a channel and a time range. When waveform (`Adc` or `RealWave`) data is read, the time of the first waveform data point is returned, as well as the data itself. If there is a gap in the waveform data within the requested range, data is only returned up to the gap. The following data must be accessed by a new read request, timed to start after the end of the data from the previous read.

For all channel types, if the buffer is returned full, further reads should be done with the start time set to be the time of the last data item in the buffer plus 1, as the only way to be sure you have reached the end of the data is if you are returned a partially full buffer.

Writing data

Data is written in time sequence for a particular channel and the channels can be written in any order. Each channel forms a doubly linked list of data blocks in the file. You can read from a file while it is open for writing.

Data channels can be deleted. If this is done, the chain of blocks that was in use by a channel remain and will be re-used if the channel is re-written. The only way to remove the space used by a deleted channel is to write a program to copy the file to a new file, omitting the deleted channels.

File selection and number of files

When a SON file is opened or created, the library returns an integer value called a handle, to the calling software. All subsequent function calls that operate on this file must supply the handle. When the file is closed the file handle becomes invalid. The SON library allows up to 2048 files open at any one time, but available memory may reduce this. File description tables occupy memory space when the files are in use, so it is good practice to close files when they are no longer needed.

Physical disk storage

The data for each channel is written to disk in blocks. For a particular channel, the block is of a fixed size, although the block need not be full. All blocks are a multiple of 512 bytes long. Each block has a header that is 20 bytes long. This header holds the pointers to the previous and the next block in the file, the times of the first and last data items in the block, the channel number and the number of data items in the block.

In the file header is an array of descriptors, one for each data channel. These descriptors have pointers to the first and last data blocks for each channel. We use a pointer of value -1 to mean the end of a chain of blocks.

Data is found by skipping along the linked lists until the required block is found. The library builds an index table of blocks in order to speed up the search for data. You will not normally need to use any of this physical storage information. More details are available in the chapter on internal structures and functions

Types and structures

The following data types are defined for C/C++ users of the SON library. Some are defined in the `MACHINE.H` include file that attempts to compensate for machine and environment difference, most are defined in the `SON.H` definition of the SON library:

Simple types

```
typedef unsigned char BOOLEAN;
typedef unsigned short WORD;
typedef unsigned long DWORD;
typedef short TAdc;
typedef TAdc FAR * TpAdc;
typedef long TSTime;
typedef TSTime FAR *TpSTime;
typedef char FAR * TpStr;
typedef const char FAR * TpCStr;
typedef WORD FAR * TpWORD;
typedef BOOLEAN FAR * TpBOOL;
typedef float FAR * TpFloat;
typedef void FAR * TpVoid;
typedef short FAR * TpShort;
```

The `FAR` qualifier is a relic of 16-bit DOS and compiles away to nothing except in 16-bit DOS builds of the library.

The `TSTime` type is a formal definition of the data type used by SON to hold a time value, similarly the `TAdc` type defines an ADC value. The other types provide formal definitions of pointers to various types of data, both for use as function parameters and to ensure that `FAR` pointers are used where necessary. `FAR` is ignored where it is irrelevant (Mac and 32-bit Windows).

When a `TpCStr` argument is used to pass a text string to an exported function (comments, titles and units), the string is always terminated by a zero byte. The string can be longer or shorter than the maximum length specified. If it is longer, only characters up to the specified length are used.

When a string is read back, we use the `TpStr` type. The storage area supplied must be long enough for the maximum string length plus the zero terminating character.

TDataKind

This type defines the kind of data that is stored in each channel. The `RealWave` type was new at version 6. The `AdcMark` type was extended at version 6 to allow interleaved data.

```
typedef enum
{
    ChanOff=0,          /* the channel is OFF - */
    Adc,                 /* a 16-bit waveform channel */
    EventFall,          /* Event times (falling edges) */
    EventRise,          /* Event times (rising edges) */
    EventBoth,          /* Event times (both edges) */
    Marker,             /* Event time plus 4 8-bit codes */
    AdcMark,            /* Marker plus Adc waveform data */
    RealMark,           /* Marker plus float numbers */
    TextMark,           /* Marker plus text string */
    RealWave            /* waveform of float numbers */
} TDataKind;
```

TSONTimeDate

```
typedef struct
{
    unsigned char ucHun;   /* hundreths of a second, 0-99 */
    unsigned char ucSec;   /* seconds, 0-59 */
    unsigned char ucMin;   /* minutes, 0-59 */
    unsigned char ucHour;  /* hour - 24 hour clock, 0-23 */
    unsigned char ucDay;   /* day of month, 1-31 */
    unsigned char ucMon;   /* month of year, 1-12 */
    WORD wYear;            /* year 1980-65535! */
}TSONTimeDate;
```

TSONCreator

```
typedef struct
{
    char acID[8];
} TSONCreator;                /* application identifier */
```

Marker types

The `TMarkBytes` and `TMarker` definitions together give the definition of a Marker, a time value with 4 bytes of attached data. The `TAdcMark`, `TRealMark` and `TTextMark` types are all markers with an attached array of data, only the type of data varies. The length of the attached data array is variable; the lengths given are nominal maximum lengths. All of these structures are packed on 2-byte boundaries.

```
typedef char TMarkBytes[4]
typedef TMarkBytes FAR * TpMarkBytes;

typedef struct
{
    TSTime mark;              /* Marker time as for events */
    TMarkBytes mvals;         /* the marker values */
} TMarker;

#define SON_MAXADCMARK 1024   /* maximum points in AdcMark (arbitrary) */
#define SON_MAXAMTRACE 4      /* maximum interleaved traces in AdcMark */
typedef struct
{
    TMarker m;                /* the marker structure */
    TAdc a[SON_MAXADCMARK*SON_MAXAMTRACE]; /* the attached ADC data */
} TAdcMark;

#define SON_MAXREALMARK 512   /* maximum points in RealMark (arbitrary) */
typedef struct
{
    TMarker m;                /* the marker structure */
    float r[SON_MAXREALMARK]; /* the attached floating point data */
} TRealMark;

#define SON_MAXTEXTMARK 2048  /* maximum points in Textmark (arbitrary) */
typedef struct
{
    TMarker m;                /* the marker structure */
    char t[SON_MAXTEXTMARK];  /* the attached text data */
} TTextMark;

typedef TMarker FAR * TpMarker;
typedef TAdcMark FAR * TpAdcMark;
typedef TRealMark FAR * TpRealMark;
typedef TTextMark FAR * TpTextMark;
```

Marker filter types

These define the structure used to hold a marker filter; a definition of which markers are wanted. See below for more on marker filtering.

```
#define SON_FMASKSZ 32                       /* # of TFilterElt in mask */
typedef unsigned char TFilterElt;            /* element of a map */
typedef TFilterElt TLayerMask[SON_FMASKSZ]; /* 256 bits in the bitmap */

typedef struct
{
    long lFlags;            /* private flags used by marker filering */
    TLayerMask aMask[4];    /* set of masks for each layer */
} TFilterMask;
typedef TFilterMask FAR *TpFilterMask;


#define SON_FMASK_ORMODE 0x02000000  /* use OR if data rather than AND */
#define SON_FMASK_VALID  0x02000000  /* bits that are valid in the mask */
#define SON_FALLLAYERS  -1
#define SON_FALLITEMS   -1
#define SON_FCLEAR        0
#define SON_FSET          1
#define SON_FINVERT      2
#define SON_FREAD       -1
```

Internal library information

Internal structures and functions

In addition to the structures and functions declared in `SON.H`, there is a separate include file, `SONINTL.H`, declaring structures and functions normally only used internally. This file includes the definition of all of the header structures used in SON files and the master table containing details of all the files open, a SON file handle is an index into this table. It is not intended that any of these structures or functions should be used by programmers writing applications using SON, they are documented for completeness and for the use of systems programmers extending the SON library.

Structure on disk

File header `TFileHeader` 512 bytes
Channel table `TChannel[n]` 140 bytes per channel rounded up to multiple of 512 bytes Includes disk offset to first and last data block of this channel Number of channels is in file header
Extra data Size (may be 0) held in file header
Data blocks `TDataBlock` Each data block holds the disk offset of the next and previous data block on the same channel. The value 0xffffffff marks each end.

All son files start with a header that is 512 bytes long. This corresponds to the `TFileHead` structure described below. The header has information to identify the file and the application that wrote it, the number of data channels, the offset to the start of the data area, the size of the extra data area and the basic time base information for the file.

The channel table follows the header. This corresponds to an array of `TChannel`, with one `TChannel` structure per channel. The size of `TChannel` is 140 bytes. The channel area is rounded up in size to a multiple of 512 bytes.

The information stored for each channel includes the size of each channel data block, the number of data blocks and the disk offset of the first and last data block for the channel. Channels can be deleted, in which case the number of deleted blocks is saved in the channel block and the disk offset to the first deleted block. If a channel is reused, the deleted blocks can either be re-used, or they can be abandoned (leaving gaps in the file). The only way to remove deleted channel data is to rewrite the file.

An optional extra data area that is reserved for application specific data follows the channel information. The SON filing system provides routines to read and write this area, but has nothing to say about the structure of the extra data area. The size of the extra data area is held in the file header.

The remainder of the file is composed of data blocks and deleted data blocks. These correspond to the `TDataBlock` structure. The size of each data block is requested by the application that writes the file. The SON library rounds up the size to the next multiple of 512 bytes. Data is always arranged on 512 byte boundaries because many physical disk systems use this as a sector size (so it is efficient), and it allows us to scan damaged files to find blocks headers knowing that all blocks start on a 512 byte boundary.

Each data block has a 20 byte header that is followed by the channel data. The header has a pointer to the previous and next data block on the same channel (or 0xffffffff to mark the end of the list), the time in clock ticks of the first and last data item in the block, the number of data items in the block, the channel number, and a flag for `EventLevel` data to indicate the level of the first data item in the block. Blocks are not changed if the channel they belong to is deleted.

SONINTL.H contents

This description of the internal data file omits structures that are not part of the visible disk format of the file. Purely internal structures are maintained here because earlier versions of the filing system made them visible, but we may change them in future realeases. The only reason for making use of the information here should be in furthering your understanding of the system or to write code to read a SON file without making use of the SON library.

Simple constants

These constants and types are defined in `SONINTL.H` for use within the SON library.

```
#define LSTRING(size) union{unsigned char len;char string[size+1];}
#define REVISION 6
#define MAXFILES 32              /* Max no. of files (ignored for WIN32) */
#define MAXLOOK 512                     /* Lookup table entries per file */
#define MAXWBUF 64              /* Write buffers per channel in new file */
#define CHANGES 8              /* Stored changes per channel in new file */
#define DISKBLOCK 512                              /* Size of a disk block */
#define ROUND_TO_DB(num) (((num)+DISKBLOCK-1)&0xfe00)
#define LENCOPYRIGHT  10       /* Length of copyright and serial strings */
#define LENSERIALNUM   8
#define COPYRIGHT "(C) CED 87"              /* The copyright string used */
```

File header

The `TFileHead` structure is an image of the first 512 bytes of a `SON` file. It contains the SON file identification, general information about the file and channels and the file comment. In version 6, the serial number field (never used in anger) is replaced by the `creator` field, which is supported by `SONAppId()`. We have also added the `dTimeBase` and `timeDate` fields (previously all set to 0's in version 5).

```
typedef struct      /* first disk block of file */
{
    short systemID;               /* filing system revision level */
    char copyright[LENCOPYRIGHT]; /* space for "(C) CED 87" */
    TSONCreator creator;          /* optional application identifier */
    WORD usPerTime;               /* microsecs per time unit */
    WORD timePerADC;              /* time units per ADC interrupt */
    short fileState;              /* condition of the file */
    long firstData;               /* offset to first data block */
    short channels;               /* maximum number of channels */
    WORD chanSize;                /* memory size to hold chans */
    WORD extraData;               /* No of bytes of extra data in file */
    WORD bufferSz;                /* Not used on disk; bufferP in bytes */
    WORD osFormat ;               /* 0x0101 for Mac, or 0x0000 for PC */
    TSTime maxFTime;              /* max time in the data file */
    double dTimeBase;             /* time scale factor, normally 1.0e-6 */
    TSONTimeDate timeDate;        /* time that corresponds to tick 0 */
    char pad[52];                 /* padding for the future */
    TFileComment fileComment;     /* what user thinks of it so far */
} TFileHead;
typedef TFileHead FAR * TpFileHead;
```

When the SON library opens a file, it loads the file header into memory, then if the `system` ID is not the latest version, it upgrades the header to the latest version. When the file closes, the library checks the file contents and sets the header to the oldest version that is compatible with the data in the file so that old applications can still read files that do not use new features.

Channel header

The `TChannel` structure is an image of the channel information stored on disk after the file header. A SON file contains an array of `TChannel` structures starting immediately after the file header. The rest of the file consists of data blocks, which are forward and backwards linked into lists, one list per channel (plus an optional second list of deleted blocks). The channel structure contains pointers to the start and end of the linked list of blocks.

```
typedef struct
{
    WORD delSize;       /* number of blocks in deleted chain, 0=none */
    long nextDelBlock; /* if deleted, first block in chain pointer */
    long firstBlock;    /* points at first block in file */
    long lastBlock;     /* points at last block in file */
    WORD blocks;        /* number of blocks in file holding data */
    WORD nExtra;        /* Number of extra bytes attached to marker */
    short preTrig;      /* Pre-trig points for ADC Marker data */
    short free0;        /* Keeps space OK */
    WORD phySz;         /* physical size of block written =n*512 */
    WORD maxData;       /* maximum number of data items in block */
    TChanComm comment;  /* string commenting on this data */
    long maxChanTime;   /* last time on this channel */
    long lChanDvd;      /* waveform divide from usPerTime, 0 for others */
    short phyChan;      /* physical channel used */
    TTitle title;       /* user name for channel */
    float idealRate;    /* ideal rate:ADC, estimate:event */
    TDataKind kind;     /* data type in the channel */
    unsigned char pad;  /* padding byte */

    union                       /* Section that changes with the data */
    {
        struct
        {                       /* Data for ADC and ADCMark channels */
            float scale;
            float offset;       /* to convert to units */
            TUnits units;       /* channel units */
            WORD divide;        /* V5:ADC divide, V6:AdcMark interleave */
        } adc;
        struct
        {                       /* only used by EventBoth channels */
            BOOLEAN initLow;    /* initial event state */
            BOOLEAN nextLow;    /* expected state of next write */
        } event;
        struct
        {                       /* RealMark and RealWave */
            float min;          /* expected minimum value */
            float max;          /* expected maximum value */
            TUnits units;       /* channel units */
        } real;                 /* NB this is laid out as for adc data */
    } v;

} TChannel;
typedef TChannel FAR * TpChannel;
```

When the SON library opens a file, it reads the channel information for all channels into memory. If the file is not the latest version, the library updates the channel information to the latest standard. When upgrading to V6, the `lChanDvd` field for `Adc` and `AdcMark` channels is set to `divide`* `timePerADC` and `divide` is set to 1. When a file is closed, the channel information is down-graded to the oldest format that would not lose information.

An `Adc` channel can be set back to V5 if `lChanDvd` is divisible by `timePerADC` and the result is not greater than 65535. An `AdcMark` channel can be set back if the same condition holds and the channel interleave is 1. If the file contains any channels of type `RealWave` it cannot be down-graded to an older version.

Data block

The `TDataBlock` structure defines a data block used to hold channel data. Data blocks are arranges as doubly linked lists, with each block holding a pointer to the next and previous block in the chain. Actually these are not pointers, but offsets from the start of the file. Data blocks are of variable size, the fixed-size data types are defined as arrays of length 1 so that standard array indexing mechanisms can be used to retrieve data.

```
typedef struct
{
    long   predBlock;     /* predecessor block in the file */
    long   succBlock;     /* following block in the file */
    TSTime startTime;     /* first time in the file */
    TSTime endTime;       /* last time in the block */
    WORD   chanNumber;    /* The channel number in the block */
    WORD   items;         /* Actual number of data items found */
    union
    {
        TAdc      int2Data [ADCdataBlkSize];    /* ADC data */
        TSTime    int4Data [timeDataBlkSize];   /* time data */
        TMarker   markData [markDataBlkSize];   /* marker data */
        TAdcMark  adcMarkData;                  /* ADC marker data */
        float     realData [realDataBlkSize];   /* RealWave data */
    } data ;
} TDataBlock;
typedef TDataBlock FAR * TpDataBlock;         /* Pointer to a data block */
#define SONDBHEADSZ 20                       /* size of TDataBlock header */
```

File data

The `TFH` type is defined to be whatever type acts as a file handle so that other functions can use this type without having to care about what it is.

```
#if defined(macintosh) || defined(_MAC)
  #define   TFH int
#else
  #ifdef WIN32
    #define TFH HANDLE
  #else
    #ifdef LLIO
      #define   TFH int
    #else
      #define   TFH FILE*
    #endif
  #endif
#endif
```