SonPy

1.8.2

Generated by Doxygen 1.8.16

1 Python interface to the SON64 library

1.1 Introduction

SonPy provides a way to access the Son64 library filing system from Python, and is created with PyBind11. If you see references to "pyson", this is a hangover from a previous naming convention

1.1.1 Installation

SonPy is available for Python 3 only, for versions 3.7 through 3.9. The Linux distributable has been built for Debian, and is expected to work on most other flavours, especially Ubuntu, which is derived from Debian.

The default installer is only supplied as a wheel, but if you cannot use these for some reason, contact CED directly and we may be able to create a traditional source distributable. Note however that this doesn't actually contain any source code, it is just not a wheel!

Ideally, have numpy installed first. Then, install the latest available version of SonPy from PyPI with: "pip install sonpy". The example files also use matplotlib, which is not included by default but can be pip installed, and tkinter, which should come already packaged with relevant versions of Python.

1.1.2 Package layout

Because of cross-platform requirements, the actual SonPy library is in a sub-package, called lib. Thus to use it, you will need to "import sonpy.lib".

1.1.3 Testing

Once installed, you can test that SonPy is working by running the test scripts. These should be available with "import sonpy.example_name", and can be found in the module install directory, under site-packages/sonpy. They are not laid out for any particular use case, but are simply provided in order to show some of the objects, functions, enumerations etc. in the SonPy module.

MakeFile.py will create a 32- and 64-bit Spike2 file and add some channels containing a short amount of arbitary data.

ReadWave.py will prompt you to select an existing Spike2 file and provide a time in seconds. It will then attempt to open the file and plot that number of seconds worth of data from the first WaveForm or RealWave channel it finds. The import will fail if the amount of time you request includes any gaps (see the help for ReadWave()).

ReadFile.py will read all data in a file. You may find this useful to export entire files to Python for your own analysis, but be careful when doing so. It is almost certainly best to export only the channels you need as and when you need them, as exporting the whole file at once may take a large amount of resources.

1.1.4 Documentation

Detailed documentation is provided in the module installation directory, as well as the help that is available from within Python by calling help(sonpy), help(DigMark) etc. Because sonpy has been compiled directly from C++ code however, the separate documentation mirrors the layout of the source code, which is not Pythonic and is likely to be confusing to those without some basic knowledge of C++. In addition, we have endeavoured to make sure that all names have been maintained in the binding code, but we cannot guarantee that every name in the help pages matches the internal Python help, which will always have the correct names.

1.1.5 Contact CED

For help setting up or using SonPy, please contact the CED software helpdesk (see http://ced.co.↩uk/support/introduction).

2 Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 File Index

4.1 File List

Here is a list of all documented files with brief descriptions:

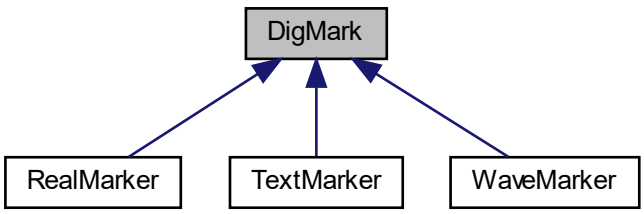
5 Class Documentation

5.1 DigMark Struct Reference

A simple digital marker.

`#include <pmarker.h>`

Inheritance diagram for DigMark:



5.1.1 Detailed Description

A simple digital marker.

A marker consists of 5 pieces of information: A 64-bit time stamp or tick value, which is measured in units of the file timebase. Four unsigned 8-bit marker codes, with which the markers can be filtered.

The documentation for this struct was generated from the following file:

- D:/hgwork/sonpy/pmarker.h

5.2 MarkerFilter Class Reference

A class for filtering markers.

`#include <pfilter.h>`

Public Member Functions

- MarkerFilter ()

  *Default constructor that makes a filter with a small **repr** output.*
- bool Filter (const DigMark &Marker) const

  See whether a Marker passes this filter.
- void SetMode (FilterMode eMode)

  Set the filter mode.
- FilterMode GetMode () const

  Query the current FilterMode.
- int SetItem (int iLayer, int iItem, FilterSet eSet)

  Tell the filter which marker codes to accept.
- int SetLayer (int iLayer, std::array< FilterSet, 256 > eSet)

  Sets all of the items in a layer.
- bool GetItem (int iLayer, int iItem) const

  Gets an item in a specified layer.
- std::vector< bool > GetLayer (int iLayer) const

  Gets either all of the items in a sepcified layer or the item in each active layer.
- FilterState GetState (int iLayer) const

  Check to see what items the filter will pass.
- void SetColumn (int nCol)

  Set the column to return when reading extended marker data.
- int GetColumn () const

  Get which column is used when returning data.
- bool operator== (const MarkerFilter &rhs) const

  Test if two filters are the same.

5.2.1 Detailed Description

A class for filtering markers.

Objects of this class are used to filter Marker-derived channels to determine if a particular data item is wanted or not. This class is also used to choose which trace AdcMark channels with multiple traces return when read as Adc data.

A new filter will pass all markers and traces by default.

A filter consists of several items, the accessible ones of which are: four bit mask layers, the filter mode and the trace column.

The mask consists of 4 layers, corresponding to the 4 possible marker codes. Each layer containing 256 items, corresponding to the possible values of each marker code. Each item can be 'Set' or 'Clear'. When the filter is applied to a marker, the Nth filter layer is searched to see if the Nth marker code is 'Set' in the layer. If it is, the filter has passed the marker.

There is also a filter mode, which can be 'First' or 'All' from the FilterMode enumeration. If set to 'First', only the first code must pass the first filter layer, whereas if set to 'All', all four codes much pass each respective layer.

Finally, there is the trace column. This is only relevant if you are reading WaveMark data as a waveform with ReadWave(). It is used to determine which column of data, or 'trace' is returned if there are multiple columns present. By default, it is unset (or rather, set to -1), which is the same as being set to zero.

There are three enumerations associated with this class, called FilterMode, FilterState and FilterSet. Each is explained by the relevant member functions. See GetMode(), SetMode(), SetItem(), GetItems() and GetState().

5.2.2 Member Function Documentation

5.2.2.1 Filter() `bool MarkerFilter::Filter ( const DigMark & Marker ) const`

See whether a Marker passes this filter.

In 'All' mode, each marker code must have the corresponding item set in the marker mask. In 'First' mode, only mask 0 is used. One of the 4 marker codes MUST be present in mask 0 with the exception that code 00 is only tested for the first marker code.

Parameters

Marker	A DigMark object to attemp to pass through the filter

Returns

Whether the marker passes the filter

5.2.2.2 GetColumn() `int MarkerFilter::GetColumn ( ) const`

Get which column is used when returning data.

Returns

The column of data that will be returned (zero-indexed). -1 Means you will get all traces.

5.2.2.3 GetItem() `bool MarkerFilter::GetItem ( int iLayer, int iItem ) const`

Gets an item in a specified layer.

Parameters

iLayer	The index of the layer to look in
iItem	The item in the layer to look at

Returns

True if the item is set, False if not

5.2.2.4 GetLayer() `std::vector< bool > MarkerFilter::GetLayer ( int iLayer ) const`

Gets either all of the items in a sepcified layer or the item in each active layer.

Parameters

iLayer	The index of the layer to query

Returns

A list of the relevant items, or containing error codes.

5.2.2.5 GetMode() `FilterMode MarkerFilter::GetMode ( ) const`

Query the current FilterMode.

Returns

A member of FilterMode, 'All' or 'First'

5.2.2.6 GetState() `FilterState MarkerFilter::GetState ( int iLayer ) const`

Check to see what items the filter will pass.

This is easy if the State is explicitly set, but if it is unset an internal calculation is performed to see what effect the filter has at the current time.

Parameters

iLayer	The layer to test, ignoring FilterMode. If -1, tests all layers using FilterMode instead.

Returns

The effect of applying this filter as a FilterState. One of All, None or Some.

5.2.2.7 operator==() `bool MarkerFilter::operator== ( const MarkerFilter & rhs ) const`

Test if two filters are the same.

Checks the number of layers, filter mode and the masks, but not the filter state.

Parameters

rhs	The filter to compare the current against

Returns

If the two filters are equivalent

5.2.2.8 SetColumn() `void MarkerFilter::SetColumn ( int nCol )`

Set the column to return when reading extended marker data.

For WaveMark channels, this means the number of the trace that is used.

Parameters

nCol	The column number to use (zero-indexed). Setting a non-existent column will default to using 0. Use -1 to view all at once.

5.2.2.9 SetItem() `int MarkerFilter::SetItem (`

```
int iLayer,
int iItem,
FilterSet eSet )
```

Tell the filter which marker codes to accept.

Items in each mask layer are either Set or Clear. You can also invert the current state.

Parameters

iLayer	The layer number, 0 to 3, or -1 for all layers
iItem	The item number in the layer, 0 to 256, or -1 for all items
eSet	A member of FilterSet: 'Clear', 'Invert' or 'Set'

Returns

An error code, zero for success

5.2.2.10 SetLayer() `int MarkerFilter::SetLayer (`

```
int iLayer,
std::array< FilterSet, 256 > eSet )
```

Sets all of the items in a layer.

Parameters

iLayer	The layer to set (0-3)
eSet	An array of FilterSet objects exactly 256 items long

Returns

An error code, zero for success

5.2.2.11 SetMode() `void MarkerFilter::SetMode (`

```
FilterMode eMode )
```

Set the filter mode.

Tell the filter whether markers need to pass all of the mask layers or only the first one.

Parameters

eMode	A member of FilterMode, 'All' or 'First'

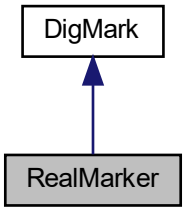
The documentation for this class was generated from the following files:

- D:/hgwork/sonpy/pfilter.h
- D:/hgwork/sonpy/pfilter.cpp

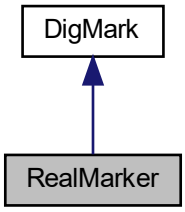
5.3 RealMarker Struct Reference

`#include <prealmark.h>`

Inheritance diagram for RealMarker:



Collaboration diagram for RealMarker:



Public Member Functions

- RealMarker (size_t nSize, int64_t nTick=0, uint8_t nCode1=0, uint8_t nCode2=0, uint8_t nCode3=0, uint8_t nCode4=0)

  Create a RealMarker with empty data of a known length and the marker codes and tick value.
- RealMarker (size_t nSize, const DigMark &mark)

  Create a RealMarker with empty data of a known length and an existing DigMark.
- RealMarker (std::vector< float > &vData, int64_t nTick=0, uint8_t nCode1=0, uint8_t nCode2=0, uint8_t nCode3=0, uint8_t nCode4=0)

  Create a RealMarker from existing data and the marker codes and tick value.
- RealMarker (std::vector< float > &vData, const DigMark &mark)

  Create a RealMarker from existing data and an existing DigMark.
- size_t Size () const

  Retrieve the length of the attached data.
- DigMark GetMark ()

  Retrieve the underlying marker and codes.

Public Attributes

- std::vector< float > Data

  The encapsulated data.

5.3.1 Detailed Description

The size of the array is determined on construction, and should correspond to the nRows value of a RealMark channel that you wish to submit this data to.

The access operator in Python is overloaded, so you can call mark[i] on a RealMarker to get or set the ith data value.

The documentation for this struct was generated from the following file:

- D:/hgwork/sonpy/prealmark.h

5.4 SonFile Class Reference

Wraps either a 64- or 32-bit File and passes function calls to it in a tidy way.

`#include <psonfile.h>`

Public Member Functions

- SonFile (std::string &sName, bool bReadOnly, OpenFlags flags=OpenFlags::None)

  Constructor for opening an existing file.
- SonFile (std::string &sName, uint16_t nChans=32, uint32_t nArea=0)

  Constructor for creating a new file.
- int GetOpenError () const noexcept

  Return any error code from an attempt to open or create a file.
- bool isFile64 () const noexcept

  Is the file 64-bit?
- bool isFile32 () const noexcept

  Is the file 32-bit?
- std::string GetName () const noexcept

  Retrieve the name of file corrently owned by this object.
- bool CanWrite () const

  Is the file read only?;.
- int EmptyFile ()

  Mark the file as empty, ready to start sampling again.
- int GetFreeChannel () const

  Get the number of the lowest channel that is Off.
- int Commit (CommitFlags flags=CommitFlags::None)

  Makes sure all data in memory is passed to the disk system.
- bool IsModified () const

  Is there unwritten data, header or channel data, in memory?
- int FlushSysBuffers ()

  Tell the disk system to move data in disk buffers to the disk.

- double GetTimeBase () const

  Retrieves the current file timebase.
- void SetTimeBase (double dSecPerTick)

  Set a new timebase.
- template<typename T >
  std::vector< T > GetExtraData (uint32_t nItems, uint32_t nOffset)

  Retrieve data from the user defined area.
- template<typename T >
  int SetExtraData (const std::vector< T > &aData, uint32_t nOffset)

  Save the binary representation of an array in the user defined area.
- uint32_t GetExtraDataSize () const

  Retrieves the size of the user defined extra data region.
- std::string GetFileComment (int n)

  Retrieves a file comment.
- int SetFileComment (int n, const std::string &sComment)

  Sets a file comment.
- int MaxChannels () const

  How many channels can this file contain?
- std::string GetAppID () const

  Retrieves the AppID.
- int SetAppID (const std::string &sData)

  Sets the AppID.
- std::vector< uint16_t > GetTimeDate ()

  Gets the saved TimeDate.
- int SetTimeDate (py::list lData)

  Sets the time and date of the file.
- int GetVersion () const

  Get the file version.
- uint64_t FileSize () const

  Get the offset to the next block the file would allocate.
- uint64_t ChannelBytes (uint16_t chan) const

  Get an estimate of how many bytes are stored for a channel.
- int64_t MaxTime (bool bReadChans=true) const

  Get the maximum time of any item in the file.
- void ExtendMaxTime (int64_t t)

  Extend or cancel the max time in a file.
- std::string GetChannelComment (uint16_t chan)

  Retrieves the channel comment if present.
- int SetChannelComment (uint16_t chan, std::string sComment)

  Save a comment for an individual channel.
- std::string GetChannelTitle (uint16_t chan)

  Retrieves the channel title.
- int SetChannelTitle (uint16_t chan, std::string sTitle)

  Set a channel title.
- std::string GetChannelUnits (uint16_t chan)

  Retrieves the channel units.
- int SetChannelUnits (uint16_t chan, std::string sUnits)

  Set the channel units.
- double GetIdealRate (uint16_t chan) const

  Get the ideal rate of a channel.
- int SetIdealRate (uint16_t chan, double dRate)

Set a new target channel rate.

- double GetChannelScale (uint16_t chan) const

  Retrieves the channel scale.
- int SetChannelScale (uint16_t chan, double dScale)

  Set the Y scale for a channel.
- double GetChannelOffset (uint16_t chan) const

  Retrieves the channel offset.
- int SetChannelOffset (uint16_t chan, double dOffset)

  Set the Y offset for a channel.
- std::pair< double, double > GetChannelYRange (uint16_t chan)

  Retrieve suggested channel limits on the Y axis.
- int SetChannelYRange (uint16_t chan, double dLow, double dHigh)

  Set the suggested Y range for a channel.
- DataType ChannelType (uint16_t chan) const

  What kind of channel is this?
- int ItemSize (uint16_t chan) const

  Get the size of the repeating item held by this channel, in bytes.
- int64_t ChannelDivide (uint16_t chan) const

  Get the waveform sample interval in clock ticks.
- int64_t ChannelMaxTime (uint16_t chan) const

  Get the time of the last item in the channel.
- int64_t PreviousNTime (uint16_t chan, int64_t trStart, int64_t trEnd=0, uint32_t n=1, bool bAsWave=false, const MarkerFilter &Filter=DefFilter()) const

  Search backwards to find the Nth point before a given time.
- int PhysicalChannel (uint16_t chan) const

  Get the physical channel number associated with a channel.
- int ChannelDelete (uint16_t chan)

  Delete a channel from the file.
- int ChannelUndelete (uint16_t chan, bool bRestore=false)

  Undelete a channel.
- void Save (int chan, int64_t t, bool bSave)

  Toggle the save state for a buffered channel.
- void SaveRange (int chan, int64_t tFrom, int64_t tUpto)

  Set a time range to be saved.
- bool IsSaving (uint16_t chan, int64_t tAt) const

  Report the save state at a certain time.
- std::vector< int64_t > NoSaveList (uint16_t chan, size_t nMax, int64_t tFrom=-1, int64_t tUpto=TSTIME64↩_MAX)

  Get a list of times where saving is turned off and on.
- double SetBuffering (int chan, size_t nBytes, double dSeconds=0.) const

  Set the buffering used with one or all channels.
- int LatestTime (int chan, int64_t t)

  Tell circular buffering the latest time we have reached during data sampling.
- int SetEventChannel (uint16_t chan, double dRate, DataType evtKind=DataType::EventFall, int iPhysChan=-1)

  Designate a channel to hold event data.
- int SetInitialLevel (uint16_t chan, bool bLevel)

  Set the initial level of a DataType::EventBoth channel.
- int WriteEvents (uint16_t chan, std::vector< int64_t > aData)

  Writes events to an event channel.

- std::vector< int64_t > ReadEvents (uint16_t chan, int nMax, int64_t tFrom, int64_t tUpto=TSTIME64_MAX, const MarkerFilter &Filter=DefFilter())

  Reads events from the specified channel.
- int SetMarkerChannel (uint16_t chan, double dRate, int iPhysChan=-1)

  Designate a channel to hold markers.
- int WriteMarkers (uint16_t chan, const std::vector< DigMark > &aData)

  Write a series of simple markers.
- std::vector< DigMark > ReadMarkers (uint16_t chan, int nMax, int64_t tFrom, int64_t tUpto, const MarkerFilter &Filter=DefFilter())

  Reads markers from marker or extended marker channels.
- int EditMarker (uint16_t chan, DigMark &Mark)

  For editing marker codes in any Marker based channel.
- int SetWaveMarkChannel (uint16_t chan, double dRate, size_t nRows, size_t nCols, int iPhysChan=-1, uint64_t tSubDvd=0, int nPre=0)

  Designate a channel to hold WaveMark data.
- int WriteWaveMarks (uint16_t chan, const std::vector< WaveMarker > &aData)

  Write WaveMarks to the file.
- std::vector< WaveMarker > ReadWaveMarks (uint16_t chan, int nMax, int64_t tFrom, int64_t tUpto=TSTI↩ ME64_MAX, const MarkerFilter &Filter=DefFilter())

  Read WaveMark data.
- int EditWaveMark (uint16_t chan, WaveMarker &Mark)

  Edit a WaveMark.
- int SetRealMarkChannel (uint16_t chan, double dRate, size_t nRows, int iPhysChan=-1)

  Designate a channel to hold RealMark data.
- int WriteRealMarks (uint16_t chan, const std::vector< RealMarker > &aData)

  Write RealMarks to the file.
- std::vector< RealMarker > ReadRealMarks (uint16_t chan, int nMax, int64_t tFrom, int64_t tUpto=TSTIM↩ E64_MAX, const MarkerFilter &Filter=DefFilter())

  Read RealMark data.
- int EditRealMark (uint16_t chan, RealMarker &Mark)

  Edit a RealMark.
- int SetTextMarkChannel (uint16_t chan, double dRate, size_t nMax, int iPhysChan=-1)

  Designate a channel to hold TextMark data.
- int WriteTextMarks (uint16_t chan, const std::vector< TextMarker > &aData)

  Write TextMarks to the file.
- std::vector< TextMarker > ReadTextMarks (uint16_t chan, int nMax, int64_t tFrom, int64_t tUpto=TSTIM↩ E64_MAX, const MarkerFilter &Filter=DefFilter())

  Read TextMark data.
- int EditTextMark (uint16_t chan, TextMarker &Mark)

  Edit a TextMark.
- std::vector< size_t > GetExMarkInfo (uint16_t chan)

  Get information on extended marker data.
- int SetWaveChannel (uint16_t chan, int64_t tDivide, DataType eKind, double dRate=0., int iPhysChan=-1)

  Designate a channel to hold Wavemark or RealWave data.
- int64_t WriteInts (uint16_t chan, std::vector< short > aData, int64_t tFrom)

  Write int data to an Adc channel.
- int64_t WriteFloats (uint16_t chan, std::vector< float > aData, int64_t tFrom)

  Write float data to a RealWave channel.
- template<typename T >
  std::vector< T > ReadWave (uint16_t chan, int nMax, int64_t tFrom, int64_t tUpto, const MarkerFilter &Filter)

  Reads a wave and returns it in an array of ints or floats.
- int64_t FirstTime (uint16_t chan, int64_t tFrom, int64_t tUpto, const MarkerFilter &Filter=DefFilter())

  Search forward to find the point after a given time.

5.4.1 Detailed Description

Wraps either a 64- or 32-bit File and passes function calls to it in a tidy way.

This class is meant to obey RAII, in that when you create an instance of one you must either create or open a file, and when you delete one the resources are closed and saved. The only time this is not the case is when there is a construction error, and you will be left with an empty SonFile owning no resources, just an error code.

There are several cases where the underlying functions require pointers to be passed. To avoid this in Python, we either ask for lists/arrays and use the data in them, or return a newly created array or list. As the internal functions in C++ do not benefit from the useful Python feature of have list 'views', the more times we have to manipulate data to get it into a list the more copying is involved. Reading extended markers are particularly bad cases, as we return a list of two arrays.

In some cases, most notably ReadWave(), we ask for an array by reference and append/edit it as neccessary. This has two advantages: firstly, we can return an error code (none of this code throws any custom exceptions) separately to the data we return, but more useful is that we can perform basic type checking on the array given, and proceeed accordingly (there are two copies of WriteWave() in the native library, one each for float and short data).

The family of read functions are hierarchical, in that a function of 'lower' complexity can be successfully called on channels of 'higher' complexity. For instance, ReadEvents() can be called on marker or extended marker channels, but ReadMarkers() cannot be called on event channels.

There are two constructors, one for opening files which requires at least two explicit parameters and another for file creation, which only explicitely requires a file name.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 SonFile() [1/2]

```
SonFile::SonFile (
        std::string & sName,
        bool bReadOnly,
        OpenFlags flags = OpenFlags::None )
```

Constructor for opening an existing file.

Whether we try and open the file as 32- or 64-bit depends on the file extension. ".smrx" gets opened as 64-bit, andyhting else is treated as 32-bit. If we succeed in opening the file, m_iOpenErr is set to S64_OK, else an appropriate error.

Parameters

sName	The path of the existing file to open
bReadOnly	If the file will be opened in read only mode
flags	One of the OpenFlags enum

5.4.2.2 SonFile() [2/2]

```
SonFile::SonFile (
        std::string & sName,
```

```
        uint16_t nChans = 32,
        uint32_t nArea = 0 )
```

Constructor for creating a new file.

Creates a new SON file. If we succeed, m_iOpenErr is set to S64_OK, else an appropriate error.

In the case that the file is 64-bit, some preparation is handled automatically, such as initiallising buffering for channel writing, and setting a default timebase of 1 microsecond.

Using wExtra, you can set aside an area for entirely arbitary purposes. Currently, we make no use of this, so Spike2 will completely ignore any data in this section. This area is limited to 65536 bytes.

Parameters

sName	The name of the new file. We use this to determine architecture. If the name ends with ".smr" we create a 32-bit file, else it is 64-bit.
nChans	The desired number of channels (minimum 32)
nArea	The desired extra space in the file header, in bytes.

5.4.3 Member Function Documentation

5.4.3.1 ChannelBytes() `uint64_t SonFile::ChannelBytes (`

```
        uint16_t chan ) const
```

Get an estimate of how many bytes are stored for a channel.

This is really only to say if there is any data stored or not for the channel and maybe to allow an estimate of the space/time needed for an operation on the channel. This will be reasonably accurate as long as all buffers written are full except the last. If you are sampling and have circular buffers, uncommitted data in the circular buffers is included.

To get an upper estimate of the number of data items held by the channel, divide the returned size by the channel ItemSize().

Parameters

chan	The channel to query

Returns

Bytes on or commited to the disk for the channel

5.4.3.2 ChannelDelete() `int SonFile::ChannelDelete (`

```
        uint16_t chan )
```

Delete a channel from the file.

In a 64-bit file, channels are deleted in such a way that as long as you do not reuse the channel, it is possible to undelete them. To attempt this, see ChannelUndelete().

Parameters

chan	The channel to delete

Returns

Error code, zero for success

5.4.3.3 ChannelDivide() `int64_t SonFile::ChannelDivide ( uint16_t chan ) const`

Get the waveform sample interval in clock ticks.

This is used by channels that sample equal interval waveforms and is the number of file clocks ticks per sample.

Parameters

chan	The channel to query

Returns

The sample interval in clock ticks, or 1 if the channel doesn't exist

5.4.3.4 ChannelMaxTime() `int64_t SonFile::ChannelMaxTime ( uint16_t chan ) const`

Get the time of the last item in the channel.

This works for both channels read from disk and channels that are being written.

Parameters

chan	The channel to query

Returns

The time of the last item in the channel, -1 if there is no data, or an error code.

5.4.3.5 ChannelType() `DataType SonFile::ChannelType ( uint16_t chan ) const`

What kind of channel is this?

Returns

The channel type, which can be Off. See the DataType enumeration.

5.4.3.6 ChannelUndelete()

```
int SonFile::ChannelUndelete (
        uint16_t chan,
        bool bRestore = false )
```

Undelete a channel.

To see if you can undelete a channel, call this first with bRestore as false (the default). If the result is not ChanOff, you can attempt to undelete the channel by calling again with bRestore as true.

Parameters

chan	The channel to undelete
bRestore	If false, requests the type of the the deleted channel. If true, attempts to undelete it.

Returns

If bRestore is false, the channel kind. Otherwise, an error code, zero for success.

5.4.3.7 Commit()

```
int SonFile::Commit (
        CommitFlags flags = CommitFlags::None )
```

Makes sure all data in memory is passed to the disk system.

Write any parts of the file that need writing. Note that this just means that the data has made it as far as the operating system buffers. The default flags value of 0 will make sure that all dirty data (in the file header or channel data) is written to the operating system. The operating system will likely have its own disk buffers and it is entirely up to the operating system to decide when data is physically transferred to the disk. Even when data is sent to the disk, this does not guarantee permanent storage as disks have their own buffering systems which allow them to reorder writes into the most efficient (least head moving) order.

Be very careful and make sure you understand how the flags parameter works if you wish to depart from the default behaviour.

Parameters

flags	one of the CommitFlags items

Returns

Error code, zero for success

5.4.3.8 EditMarker()

```
int SonFile::EditMarker (
        uint16_t chan,
        DigMark & Marker )
```

For editing marker codes in any Marker based channel.

Does not currently allow editing the time, only the following codes.

Parameters

chan	The channel to edit in
Marker	A DigMark object, which replaces an existing marker in this channel. Marker must have the same tick value as the marker to be replaced.

Returns

Error code, zero for success

5.4.3.9 EditRealMark()

```
int SonFile::EditRealMark (
        uint16_t chan,
        RealMarker & Mark )
```

Edit a RealMark.

Parameters

chan	The channel number
Mark	A RealMarker which replaces an existing one in this channel. It must have the same tick value and length of data as the marker to be replaced.

Returns

Error code, zero for success

5.4.3.10 EditTextMark()

```
int SonFile::EditTextMark (
        uint16_t chan,
        TextMarker & Mark )
```

Edit a TextMark.

Parameters

chan	The channel number
Mark	A TextMarker which replaces an existing one in this channel. The length of the string must not exceed the maximum allowed for this channel. See GetExMarkInfo().

Returns

Error code, zero for success

5.4.3.11 EditWaveMark() `int SonFile::EditWaveMark (`

```
uint16_t chan,
WaveMarker & Mark )
```

Edit a WaveMark.

Parameters

chan	The channel number
Mark	A WaveMarker which replaces an existing one in this channel. It must have the same tick value and length of data as the marker to be replaced.

Returns

Error code, zero for success

5.4.3.12 EmptyFile() `int SonFile::EmptyFile ( )`

Mark the file as empty, ready to start sampling again.

Sets the channel pointers back to time = 0, but doesn't change the settings or overwrite any data, so we can be quick.

Returns

Error code, zero for success

5.4.3.13 ExtendMaxTime() `void SonFile::ExtendMaxTime (`

```
int64_t t )
```

Extend or cancel the max time in a file.

MaxTime() can find the maximum time in the file by scanning all the channels. However, each time we write data we update the file head to hold the last written time. This can be used to make a file appear longer than it is, so we may wish to reset the last time.

Parameters

t	Set the MaxTime to this. -1 effectively removes MaxTime, and a value larger than MaxTime(false) extends it.

5.4.3.14 FileSize() `uint64_t SonFile::FileSize ( ) const`

Get the offset to the next block the file would allocate.

When a file is opened for reading, the physical file size should be the same or a small amount less than this. If it is more, the file has extra information written on the end and needs fixing. It was probably interrupted during writing. This does not included buffered data that is not committed to disk during writing.

Returns

The (estimated) file size in bytes, usually a multiple of 64 kB.

5.4.3.15 FirstTime()

```
int64_t SonFile::FirstTime (
        uint16_t chan,
        int64_t tFrom,
        int64_t tUpto,
        const MarkerFilter & Filter = DefFilter() )
```

Search forward to find the point after a given time.

WaveMark channels can be read provided there is suitable time axis information.

Parameters

chan	The channel number
tFrom	Time to start searching from
tUpto	Time to stop searching at
Filter	An optional MarkerFilter to filter the results by if the channel is AdcMark

Returns

The time of the first item in the specified time range or an error code.

5.4.3.16 FlushSysBuffers() `int SonFile::FlushSysBuffers ( )`

Tell the disk system to move data in disk buffers to the disk.

Where supported, this tells the OS to make sure that data written to OS buffers ends up on disk. This can be very inefficient as it is allowed to remove all data from disk caches, resulting in very slow operation until data is reloaded.

Returns

Error code, zero for success

5.4.3.17 GetAppID() `std::string SonFile::GetAppID ( ) const`

Retrieves the AppID.

Returns

The AppID, or a string corresponding to an error code

5.4.3.18 GetChannelComment() `std::string SonFile::GetChannelComment ( uint16_t chan )`

Retrieves the channel comment if present.

Parameters

chan	Channel number

Returns

The channel comment, or a string corresponding to an error code

5.4.3.19 GetChannelOffset() `double SonFile::GetChannelOffset ( uint16_t chan ) const`

Retrieves the channel offset.

See SetChannelScale().

Parameters

chan	The channel to query

Returns

Error code, zero for success

5.4.3.20 GetChannelScale() `double SonFile::GetChannelScale ( uint16_t chan ) const`

Retrieves the channel scale.

See SetChannelScale().

Parameters

chan	The channel to query

Returns

Error code, zero for success

5.4.3.21 GetChannelTitle() `std::string SonFile::GetChannelTitle ( uint16_t chan )`

Retrieves the channel title.

Parameters

chan	Channel number

Returns

The channel title, or a string corresponding to an error code

5.4.3.22 GetChannelUnits() `std::string SonFile::GetChannelUnits ( uint16_t chan )`

Retrieves the channel units.

Parameters

chan	The channel to query

Returns

Error code, zero for success

5.4.3.23 GetChannelYRange() `std::pair< double, double > SonFile::GetChannelYRange ( uint16_t chan )`

Retrieve suggested channel limits on the Y axis.

Parameters

chan	The channel to query

Returns

An array containing the suggested low/high Y values, or an array containing an error code

5.4.3.24 GetExMarkInfo() `std::vector< size_t > SonFile::GetExMarkInfo ( uint16_t chan )`

Get information on extended marker data.

Parameters

chan	The channel to query

Returns

An array containing either [nRows, nCols, nPre] or an error code

5.4.3.25 GetExtraData() `template<typename T > std::vector<T> SonFile::GetExtraData ( uint32_t nItems, uint32_t nOffset ) [inline]`

Retrieve data from the user defined area.

This function calls a template which has 11 instances, with names like GetExtraDataChar, GetExtraDataSingle, GetExtraDataUInt16, GetExtraDataInt8 etc. The help() function in Python is not good at showing this, as it shows all int types as 'int' and all float types as 'float'.

You should only read data from this area as the type that you saved it as with SetExtraData().

Parameters

nOffset	The offset to the point to read from, in bytes
nItems	The number of data items to read

Returns

A 1D array containing either the data read or an error code

5.4.3.26 GetExtraDataSize() `uint32_t SonFile::GetExtraDataSize ( ) const`

Retrieves the size of the user defined extra data region.

Returns

The size of the user area in bytes

5.4.3.27 GetFileComment() `std::string SonFile::GetFileComment ( int n )`

Retrieves a file comment.

Parameters

n	The comment number, can be 0 to 7

Returns

The string stored in comment n

5.4.3.28 GetFreeChannel() `int SonFile::GetFreeChannel ( ) const`

Get the number of the lowest channel that is Off.

Returns

Index of next free channel (zero-indexed)

5.4.3.29 GetIdealRate() `double SonFile::GetIdealRate ( uint16_t chan ) const`

Get the ideal rate of a channel.

Parameters

chan	The channel to query

Returns

The ideal channel rate, or 0 if the channel doesn't exist

5.4.3.30 GetTimeBase() `double SonFile::GetTimeBase ( ) const`

Retrieves the current file timebase.

Everything in the file is quantified by the underlying clock tick (64-bit). As all values in the file are stored, set and returned in ticks, you need to read this value to interpret times in seconds.

Returns

Timebase in seconds

5.4.3.31 GetTimeDate() `std::vector< uint16_t > SonFile::GetTimeDate ( )`

Gets the saved TimeDate.

Returns

The stored system time, or an array containing an error code. In Spike2, this is set to the time sampling started.

5.4.3.32 GetVersion() `int SonFile::GetVersion ( ) const`

Get the file version.

Returns

The data file version or an error code

5.4.3.33 IsModified() `bool SonFile::IsModified ( ) const`

Is there unwritten data, header or channel data, in memory?

If an open data file is modified, it can have unwritten data in the file head and also in the data channels. Unwritten data can be written to the file using Commit().

Returns

True if unwritten data present else false

5.4.3.34 IsSaving() `bool SonFile::IsSaving ( uint16_t chan, int64_t tAt ) const`

Report the save state at a certain time.

See Save().

We can only say false if the data is in the circular buffer and is marked for non-saving. Once data is too old to be in the circular buffer we assume it is saved as if it was not, it will fail to be read from disk.

Parameters

chan	The channel to query
tAt	The time to check the save state at

Returns

True if the state is 'save' or the channel is non-buffered. False if it is 'don't save' or there is an error

5.4.3.35 ItemSize() `int SonFile::ItemSize ( uint16_t chan ) const`

Get the size of the repeating item held by this channel, in bytes.

Each channel holds repeating objects, all the same size. Note that all extended marker types have sizes that are rounded up to a multiple of 8 bytes. This routine is a convenient way to get the object sizes (though you could calculate them).

Parameters

chan	The channel to query

Returns

Error code, zero for success

5.4.3.36 LatestTime() `int SonFile::LatestTime ( int chan, int64_t t )`

Tell circular buffering the latest time we have reached during data sampling.

With circular buffering in use on a channel, if no data appears for a long time and no Commit() commands are issued, the commands to save and not save data ranges acumulate. By telling a channel the latest time for which no new data can be expected, the channel can clean up the save/no save list by deleting/amalgamating ranges that have no data in them.

Parameters

chan	The channel number
t	The time we have reached

Returns

Error code, zero for success

5.4.3.37 MaxChannels() `int SonFile::MaxChannels ( ) const`

How many channels can this file contain?

Returns

The number of channels the file can have

5.4.3.38 MaxTime()

```
int64_t SonFile::MaxTime (
        bool bReadChans = true ) const
```

Get the maximum time of any item in the file.

The maximum time written to disk is saved in the file head. If this is not set, the maximum time is found by scanning all the channels for the maximum time saved in any channel. We can also force all channels to be scanned with the bReadChans argument.

Parameters

bReadChans	If this is true, if any channel has a later time than the file head, the channel time is used instead. If false, and the file head holds a time, this is used.

Returns

The maximum time in the file, in ticks. If file is empty, this may be -1

5.4.3.39 NoSaveList()

```
std::vector< int64_t > SonFile::NoSaveList (
        uint16_t chan,
        size_t nMax,
        int64_t tFrom = -1,
        int64_t tUpto = TSTIME64_MAX )
```

Get a list of times where saving is turned off and on.

See Save()

Parameters

chan	The channel to query
nMax	The max numbre of changes to return. 0 for all
tFrom	Start time to search from
tUpto	End time to search to

Returns

An array of times when the save state is saved. The save state is always assumed to start as save, so the first point is always a turning off of the save state.

5.4.3.40 PhysicalChannel()

```
int SonFile::PhysicalChannel (
        uint16_t chan ) const
```

Get the physical channel number associated with a channel.

The pyhsical channel number is purely cosmetic and is not used for anything.

Parameters

chan	The channel to query

Returns

The number specified when the channel was created, -1 if unset

5.4.3.41 PreviousNTime()

```
int64_t SonFile::PreviousNTime (
        uint16_t chan,
        int64_t trStart,
        int64_t trEnd = 0,
        uint32_t n = 1,
        bool bAsWave = false,
        const MarkerFilter & Filter = DefFilter() ) const
```

Search backwards to find the Nth point before a given time.

Parameters

chan	The channel to query
trStart	Reverse start, i.e. the time to search backwards from
trEnd	Reverse end, i.e. the time to search backwards until
n	How many points to step back by
bAsWave	Use true with extended marker channels to treat marker data as a wave
Filter	A MarkerFilter to filter Marker channels

Returns

Error code, zero for success

5.4.3.42 ReadEvents()

```
std::vector< int64_t > SonFile::ReadEvents (
        uint16_t chan,
        int nMax,
        int64_t tFrom,
        int64_t tUpto = TSTIME64_MAX,
        const MarkerFilter & Filter = DefFilter() )
```

Reads events from the specified channel.

Parameters

chan	The channel number
nMax	The maximum number of events to return
tFrom	The time to start searching from
tUpto	The time so stop searching at
Filter	An optional MarkerFilter to filter the results by

Returns

An array conatining either the times of the events found or an error code

5.4.3.43 ReadMarkers()

```
std::vector< DigMark > SonFile::ReadMarkers (
    uint16_t chan,
    int nMax,
    int64_t tFrom,
    int64_t tUpto,
    const MarkerFilter & Filter = DefFilter() )
```

Reads markers from marker or extended marker channels.

Parameters

chan	The channel to read from
nMax	The max number of markers to read
tFrom	The time to start searching from
tUpto	The time so stop searching at
Filter	An optional MarkerFilter to filter the results by

Returns

A list containing either DigitalMarkers or an error code

5.4.3.44 ReadRealMarks()

```
std::vector< RealMarker > SonFile::ReadRealMarks (
    uint16_t chan,
    int nMax,
    int64_t tFrom,
    int64_t tUpto = TSTIME64_MAX,
    const MarkerFilter & Filter = DefFilter() )
```

Read RealMark data.

Parameters

chan	The channel number
nMax	The max number of marks to read
tFrom	The time to start searching from
tUpto	The time to stop searching at
Filter	An optional MarkerFilter to filter the results by

Returns

An array containing either the WaveMarkers read, or a single RealMarker where the tick value is an error code

5.4.3.45 ReadTextMarks()

```
std::vector< TextMarker > SonFile::ReadTextMarks (
        uint16_t chan,
        int nMax,
        int64_t tFrom,
        int64_t tUpto = TSTIME64_MAX,
        const MarkerFilter & Filter = DefFilter() )
```

Read TextMark data.

Parameters

chan	The channel number
nMax	The max number of marks to read
tFrom	The time to start searching from
tUpto	The time to stop searching at
Filter	An optional MarkerFilter to filter the results by

Returns

An array containing either the WaveMarkers read, or a single TextMarker where the string describes an error

5.4.3.46 ReadWave()

```
template<typename T >
std::vector<T> SonFile::ReadWave (
        uint16_t chan,
        int nMax,
        int64_t tFrom,
        int64_t tUpto,
        const MarkerFilter & Filter )  [inline]
```

Reads a wave and returns it in an array of ints or floats.

This template function can be called either with ReadInts or ReadFloats, to use the channel scale and offset to transform the data into the return type that you want (usually, this will be float).

WaveMark channels can be read provided there is suitable time axis information.

If you have been writing data sensibly, you will know the time of each of these data points, as they will be equally spaced at known multiples of the channel divider. If this is not the case, you can try to use FirstTime() to estimate the time of a single sata point.

If there are gaps in the Spike2 file, this will not be read as zero or NaN etc., it will simply be passed over, and you will end up with fewer points than the channel divide would otherwise imply. In this case, you must be extremely careful with how you treat your data.

Parameters

chan	The channel number to read from
nMax	The max number of points to read. Must not be zero.
tFrom	The time to start searching from
tUpto	The time so stop searching at
Filter	An optional MarkerFilter to filter the results by if the channel is AdcMark

Returns

An array containing the points read, or an error code

5.4.3.47 ReadWaveMarks()

```
std::vector< WaveMarker > SonFile::ReadWaveMarks (
    uint16_t chan,
    int nMax,
    int64_t tFrom,
    int64_t tUpto = TSTIME64_MAX,
    const MarkerFilter & Filter = DefFilter() )
```

Read WaveMark data.

Parameters

chan	The channel number
nMax	The max number of marks to read
tFrom	The time to start searching from
tUpto	The time to stop searching at
Filter	An optional MarkerFilter to filter the results by

Returns

An array containing either the WaveMarkers read, or a single WaveMarker where the tick value is an error code

5.4.3.48 Save()

```
void SonFile::Save (
    int chan,
    int64_t t,
    bool bSave )
```

Toggle the save state for a buffered channel.

The library saves a list of times at which save state changes will occur. See SaveList().

Parameters

chan	The channel number
t	The time from which the state set should take effect. Can be before or after the current channel max time.
bSave	True means save, false means don't save

5.4.3.49 SaveRange()

```
void SonFile::SaveRange (
        int chan,
        int64_t tFrom,
        int64_t tUpto )
```

Set a time range to be saved.

See Save().

This is expected to be used with a channel that is normally not being saved. It marks a time range for saving by adding a Save and don't save indication to the list of save state changes. You cannot change the state before the time of the last data sent to the write buffer.

Parameters

chan	The channel number
tFrom	Start of time range
tUpto	End of time range

5.4.3.50 SetAppID()

```
int SonFile::SetAppID (
        const std::string & sData )
```

Sets the AppID.

The AppID is optional, and is purely descriptive. It consists of an 8 character string.

Parameters

sData	String containing the AppID to set. Any characters after the first 8 are ignored

Returns

Error code, zero for success

5.4.3.51 SetBuffering()

```
double SonFile::SetBuffering (
        int chan,
        size_t nBytes,
        double dSeconds = 0. ) const
```

Set the buffering used with one or all channels.

Can be used for all existing channels, when it allocates buffering space and saves the buffering time, or for a single channel at a time.

If you do not call this function, no circular buffering is applied so Save() and SaveRange() will have no useful effect. In 64-bit files, you should find that this is called with some default parameters on file creation, so for simple cases you can start sampling without worrying about this.

For chan = -1: If dSeconds > 0, this is the desired buffering time. If nBytes is non-zero, it is a limit on the space to allow for all channels and will reduce dSeconds to fit with it. If nBytes = 0 and dSeconds > 0, the number of bytes is calculated and not limited. If nBytes = 0 and dSeconds <= 0, all buffering is cancelled on all channels. In all cases, the used dSeconds value is saved.

For chan > -1: If dSeconds < 0, it is replaced by any saved dSeconds. Then, if dSeconds > 0 this sets the buffering, limited by nBytes if nBytes > 0. If dSeconds = 0, the buffering is set by nBytes or removed if nBytes is 0.

Parameters

chan	The channel to set for, or -1 for all channels
nBytes	Max buffer size if non-zero
dSeconds	Desired buffering time

Returns

The buffering time set, or zero for read only files

5.4.3.52 SetChannelComment()

```
int SonFile::SetChannelComment (
        uint16_t chan,
        std::string sComment )
```

Save a comment for an individual channel.

Each channel can contain a single comment.

Parameters

chan	The channel to save the comment under
sComment	The comment to save

Returns

Error code, zero for success

5.4.3.53 SetChannelOffset()

```
int SonFile::SetChannelOffset (
        uint16_t chan,
        double dOffset )
```

Set the Y offset for a channel.

See SetChannelScale().

Parameters

chan	The channel number to set the offset of
dOffset	The new offset

Returns

Error code, zero for success

5.4.3.54 SetChannelScale()

```
int SonFile::SetChannelScale (
        uint16_t chan,
        double dScale )
```

Set the Y scale for a channel.

The channel scale and offset are used to translate between integer representations of values and real units. They are used for Adc, RealWave and AdcMark channels. When a channel is expressed in user units:

user units = integer ∗ scale / 6553.6 + offset

For a RealWave channel, where the channel is already in user units, the scale and offset values tell us how to convert the channel back into integers:

integer = (user units - offset)∗6553.6/scale

The factor of 6553.6 comes about because a scale value of 1.0 converts between an input range of 10 Volts and an ADC range of 65536. In most cases, you should not need to worry about this, just remember to use the equations.

Parameters

chan	The channel number to set the scale of
dScale	The new scale

Returns

Error code, zero for success

5.4.3.55 SetChannelTitle()

```
int SonFile::SetChannelTitle (
        uint16_t chan,
        std::string sComment )
```

Set a channel title.

Parameters

chan	The channel to title
sComment	The title

Returns

Error code, zero for success

5.4.3.56 SetChannelUnits()

```
int SonFile::SetChannelUnits (
        uint16_t chan,
        std::string sUnits )
```

Set the channel units.

Parameters

chan	The channel to set units for
sUnits	The units to set

Returns

Error code, zero for success

5.4.3.57 SetChannelYRange()

```
int SonFile::SetChannelYRange (
        uint16_t chan,
        double dLow,
        double dHigh )
```

Set the suggested Y range for a channel.

Parameters

chan	The channel to query
dLow	The suggested low limit for Y
dHigh	The suggested high limit for Y

Returns

Error code, zero for success

5.4.3.58 SetEventChannel()

```
int SonFile::SetEventChannel (
        uint16_t chan,
        double dRate,
        DataType evtKind = DataType::EventFall,
        int iPhysChan = -1 )
```

Designate a channel to hold event data.

Parameters

chan	The channel to set
dRate	The expected max event rate that may be sustained over several seconds
evtKind	Channel type, one of DataType::EventFall, DataType::EventRise or DataType::EventBoth
iPhysChan	Physical channel number associated with the channel, purely cosmetic

Returns

Error code, zero for success

5.4.3.59 SetExtraData() `template<typename T >`

```
int SonFile::SetExtraData (
            const std::vector< T > & aData,
            uint32_t nOffset )  [inline]
```

Save the binary representation of an array in the user defined area.

This function is a template, so the type that the saved data is interpreted as will be determined by the dtype of the array you pass in. Only 11 types are allowed, each of them plain old data types. These are char, single, double, and eight int types corresponding to 1, 2, 4 or 8 bytes, and signed or unsigned.

Parameters

aData	A 1D array of data to write
nOffset	How far from the start of the user defined area to start writing from, in bytes

Returns

Error code, zero for success

5.4.3.60 SetFileComment()

```
int SonFile::SetFileComment (
            int n,
            const std::string & sComment )
```

Sets a file comment.

Parameters

n	The comment number, 0 to 7
sComment	The comment to set

Returns

Error code

5.4.3.61 SetIdealRate() `int SonFile::SetIdealRate (`

```
uint16_t chan,
double dRate )
```

Set a new target channel rate.

The ideal sample rate of a channel is an exact, set value. The actual rate (from ChannelDivide()) may differ, depending on the combination of desired sample rates in other channels, and the hardware available.

Parameters

chan	Channel number to set rate for
dRate	The new rate to set, in Hertz

Returns

Error code, zero for success

5.4.3.62 SetInitialLevel() `int SonFile::SetInitialLevel (`

```
uint16_t chan,
bool bLevel )
```

Set the initial level of a DataType::EventBoth channel.

All events in an EventBoth channel are assumed to be alternating, but we need to known what the first one is. In 64-bit files, the default is rising, whereas in 32-bit files it is falling.

Parameters

chan	The channel number
bLevel	True to set the initial event as falling, False for rising

Returns

Error code, zero for success

5.4.3.63 SetMarkerChannel() `int SonFile::SetMarkerChannel (`

```
uint16_t chan,
double dRate,
int iPhysChan = -1 )
```

Designate a channel to hold markers.

Parameters

chan	The channel to set
dRate	The expected max event rate that may be sustained over several seconds
iPhysChan	Physical channel number associated with the channel, purely cosmetic

Returns

Error code, zero for success

5.4.3.64 SetRealMarkChannel()

```
int SonFile::SetRealMarkChannel (
        uint16_t chan,
        double dRate,
        size_t nRows,
        int iPhysChan = -1 )
```

Designate a channel to hold RealMark data.

Parameters

chan	The channel to set
dRate	The expected max event rate that may be sustained over several seconds
nRows	The number of rows of attached items
iPhysChan	Physical channel number associated with the channel, purely cosmetic

Returns

Error code, zero for success

5.4.3.65 SetTextMarkChannel()

```
int SonFile::SetTextMarkChannel (
        uint16_t chan,
        double dRate,
        size_t nMax,
        int iPhysChan = -1 )
```

Designate a channel to hold TextMark data.

Parameters

chan	The channel to set
dRate	The expected max event rate that may be sustained over several seconds
nMax	The maximum number of characters that can be attached
iPhysChan	Physical channel number associated with the channel, purely cosmetic

Returns

Error code, zero for success

5.4.3.66 SetTimeBase() `void SonFile::SetTimeBase (`

```
double dSecPerTick )
```

Set a new timebase.

If you change this value the file header is marked as modified and the next Commit() will save it to the file. As all times are saved in terms of ticks of this period, nothing else in the file changes, but the entire time base of the file is scaled.

Parameters

dSecPerTick	New timebase in seconds

5.4.3.67 SetTimeDate() `int SonFile::SetTimeDate (`

```
py::list lData )
```

Sets the time and date of the file.

We do not check the dtype so you must check it yourself. All values are stored as 8 bit unisgned ints, except the year, which is stored as a 16-bit unsigned int.

Parameters

lData	A list of exactly 7 values which must be interpretable as integers

Returns

Error code, zero for success

5.4.3.68 SetWaveChannel() `int SonFile::SetWaveChannel (`

```
uint16_t chan,
int64_t tDivide,
DataType eKind,
double dRate = 0.,
int iPhysChan = -1 )
```

Designate a channel to hold Wavemark or RealWave data.

Parameters

chan	The channel number

Parameters

tDivide	The spacing (divide) for the channel in timebase units; must be greater than zero
eKind	Channel kind, either Adc or RealWave
dRate	The desired sampling rate in Hz. The rate you end up with may be slightly different (see IdealRate()). If <= 0, calculated as 1/(tDivide * GetTimeBase())
iPhysChan	Physical channel number associated with the channel, purely cosmetic

Returns

Error code, zero for success

5.4.3.69 SetWaveMarkChannel()

```
int SonFile::SetWaveMarkChannel (
        uint16_t chan,
        double dRate,
        size_t nRows,
        size_t nCols,
        int iPhysChan = -1,
        uint64_t tSubDvd = 0,
        int nPre = 0 )
```

Designate a channel to hold WaveMark data.

Parameters

chan	The channel to set
dRate	The expected max event rate that may be sustained over several seconds
nRows	The number of rows of attached items
nCols	The number of columns of attached items
iPhysChan	Physical channel number associated with the channel, purely cosmetic
tSubDvd	If > 0, this implies the attached data has a time axis, and this is used as the channel divider in units of timebase, as with wave channels. To retrieve this after setting, using ChannelDivide().
nPre	If tSubDvd <= 0, this is ignored. Otherwise, it sets the index into the rows of data at which the alignment point was located.

Returns

Error code, zero for success

5.4.3.70 WriteEvents()

```
int SonFile::WriteEvents (
        uint16_t chan,
        std::vector< int64_t > aData )
```

Writes events to an event channel.

Parameters

chan	The channel number to write to
aData	A 1D array of monotonically increasing values to use as times. The dtype MUST be 64-bit signed ints, e.g. np.int64

Returns

Error code, zero for success

5.4.3.71 WriteFloats()

```
int64_t SonFile::WriteFloats (
    uint16_t chan,
    std::vector< float > aData,
    int64_t tFrom )
```

Write float data to a RealWave channel.

Unlike event-based channel types where all data must be written after any data already present, Waveform and RealWave data can be overwritten. This is to remain compatible with the 32-bit SON library but it could be argued that it should not be allowed. You cannot fill in gaps in the orignal data in this manner. It is imagined that you would use this to fix glitches or remove transients.

In normal use, all Waveform/RealWave data will be aligned at times that are at integer multiples of the channel divider. We do not prevent you writing data at other times, but programs like Spike2 may experience subtle problems if you do.

Parameters

chan	The channel number
aData	A 1D array of 2 byte ints
tFrom	The time (in file ticks) of the first point to write

Returns

The next time to write or an error code

5.4.3.72 WriteInts()

```
int64_t SonFile::WriteInts (
    uint16_t chan,
    std::vector< short > aData,
    int64_t tFrom )
```

Write int data to an Adc channel.

Unlike event-based channel types where all data must be written after any data already present, Waveform and RealWave data can be overwritten. This is to remain compatible with the 32-bit SON library but it could be argued that it should not be allowed. You cannot fill in gaps in the orignal data in this manner. It is imagined that you would use this to fix glitches or remove transients.

In normal use, all Waveform/RealWave data will be aligned at times that are at integer multiples of the channel divider. We do not prevent you writing data at other times, but programs like Spike2 may experience subtle problems if you do.

Parameters

chan	The channel number
aData	A 1D array of 4 byte floats
tFrom	The time (in file ticks) of the first point to write

Returns

The next time to write or an error code

5.4.3.73 WriteMarkers()

```
int SonFile::WriteMarkers (
    uint16_t chan,
    const std::vector< DigMark > & aData )
```

Write a series of simple markers.

Parameters

chan	The channel to write to
aData	A 1D array of DigMarks

Returns

Error code, zero for success

5.4.3.74 WriteRealMarks()

```
int SonFile::WriteRealMarks (
    uint16_t chan,
    const std::vector< RealMarker > & aData )
```

Write RealMarks to the file.

Parameters

chan	The channel number
aData	An array of RealMarkers. The data must be nRows long, where nRows can be found with GetExMarkInfo()

Returns

Error code, zero for success

5.4.3.75 WriteTextMarks() int SonFile::WriteTextMarks (uint16_t *chan*, const std::vector< TextMarker > & *aData*)

Write TextMarks to the file.

Parameters

chan	The channel number
aData	An array of TextMarkers. The maximum length of any of the strings must not exceed nRows, which can be found with GetExMarkInfo().

Returns

Error code, zero for success

5.4.3.76 WriteWaveMarks() int SonFile::WriteWaveMarks (uint16_t *chan*, const std::vector< WaveMarker > & *aData*)

Write WaveMarks to the file.

Parameters

chan	The channel number
aData	An array of WaveMarkers. The data must have dimensions of nRows x nCols. You can find these with GetExMarkInfo().

Returns

Error code, zero for success

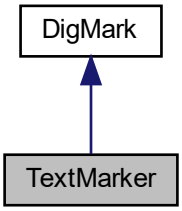
The documentation for this class was generated from the following files:

- D:/hgwork/sonpy/psonfile.h
- D:/hgwork/sonpy/psonfile.cpp

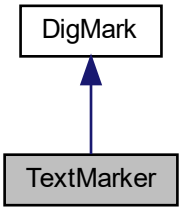
5.5 TextMarker Struct Reference

`#include <ptextmark.h>`

Inheritance diagram for TextMarker:



Collaboration diagram for TextMarker:



Public Member Functions

- TextMarker (int64_t nTick=0, uint8_t nCode1=0, uint8_t nCode2=0, uint8_t nCode3=0, uint8_t nCode4=0)

  Create a TextMarker with empty data and the marker codes and tick value.
- TextMarker (const DigMark &mark)

  Create a TextMarker with empty data and an existing DigMark.
- TextMarker (std::string &vData, int64_t nTick=0, uint8_t nCode1=0, uint8_t nCode2=0, uint8_t nCode3=0, uint8_t nCode4=0)

  Create a TextMarker from existing data and the marker codes and tick value.
- TextMarker (std::string &vData, const DigMark &mark)

  Create a TextMarker from existing data and an existing DigMark.
- size_t Size () const

  Retrieve the length of the stored string.
- DigMark GetMark ()

  Retrieve the underlying marker and codes.
- void SetString (std::string sData)
- std::string GetString () const

  Get the stored string data.

Public Attributes

- std::string Text

  The encapsulated data.

5.5.1 Detailed Description

You can store a string of any length in this object, but remember that you will get an error if you try to save it in a channel that doesn't have enough space. See GetExMarkInfo().

The access operator in Python is overloaded, so you can call mark[i] on a WaveMarker to get or set the ith char in the string. It may be prefereable however, to use the GetString() and SetString() functions.

5.5.2 Member Function Documentation

5.5.2.1 GetMark() `DigMark TextMarker::GetMark ( ) [inline]`

Retrieve the underlying marker and codes.

Set the stored string data

5.5.2.2 SetString() `void TextMarker::SetString ( std::string sData ) [inline]`

This will let you store a string of any length, so be careful that when you save a marker to a channel, the stored string contains fewer than nRows characters. See GetExMarkInfo(), Size().

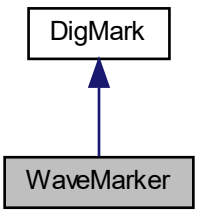
The documentation for this struct was generated from the following file:

- D:/hgwork/sonpy/ptextmark.h

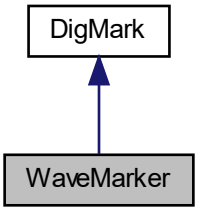
5.6 WaveMarker Struct Reference

`#include <pwavemark.h>`

Inheritance diagram for WaveMarker:



Collaboration diagram for WaveMarker:



Public Member Functions

- WaveMarker (size_t nRows, size_t nCols, int64_t nTick=0, uint8_t nCode1=0, uint8_t nCode2=0, uint8_t nCode3=0, uint8_t nCode4=0)

  Create a WaveMarker with empty data of a known length and the marker codes and tick value.
- WaveMarker (size_t nRows, size_t nCols, const DigMark &mark)

  Create a WaveMarker with empty data of a known length and an existing DigMark.
- WaveMarker (std::vector< std::vector< short > > &vData, int64_t nTick=0, uint8_t nCode1=0, uint8_t nCode2=0, uint8_t nCode3=0, uint8_t nCode4=0)

  Create a WaveMarker from existing data and the marker codes and tick value.
- WaveMarker (std::vector< std::vector< short > > &vData, const DigMark &mark)

  Create a WaveMarker from existing data and an existing DigMark.
- std::pair< size_t, size_t > Size () const

  Retrieve the length of the attached data.
- DigMark GetMark ()

  Retrieve the underlying marker and codes.

Public Attributes

- std::vector< std::vector< short > > Data

  The encapsulated data.

5.6.1 Detailed Description

The size of the array is determined on construction, and should correspond to the nRows value of a RealMark channel that you wish to submit this data to.

The access operator in Python is overloaded, so you can call mark[i,j] on a WaveMarker to set the (i, j)th data value, and you can use either mark[i,j] or mark[i][j] to get a value. Unfortunately, you cann't use mark[i][j] to set values, so for consistency, it is advised to stick to the tuple form (mark[i, j]) for both getting and setting.

The documentation for this struct was generated from the following file:

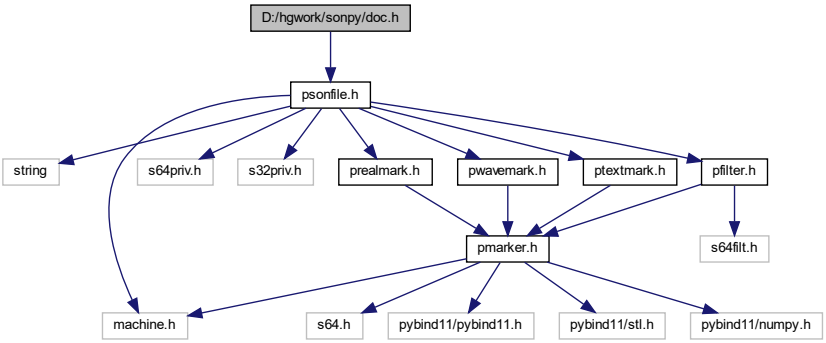
- D:/hgwork/sonpy/pwavemark.h

6 File Documentation

6.1 D:/hgwork/sonpy/doc.h File Reference

Documentation file containing no source code.

```
#include "psonfile.h"
```
Include dependency graph for doc.h:



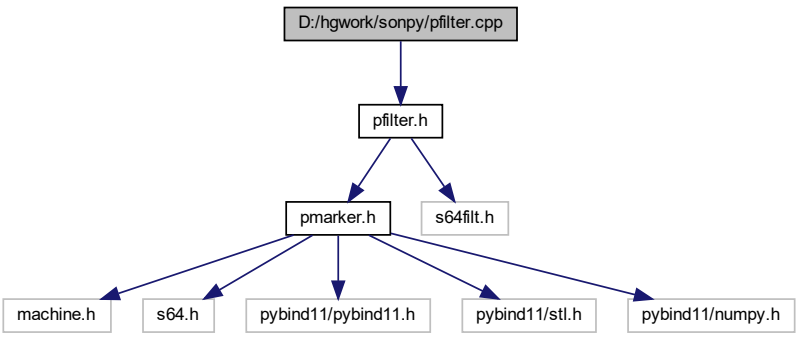
6.1.1 Detailed Description

Documentation file containing no source code.

6.2 D:/hgwork/sonpy/pfilter.cpp File Reference

Implements the MarkerFilter class.

```
#include "pfilter.h"
```
Include dependency graph for pfilter.cpp:



6.2.1 Detailed Description

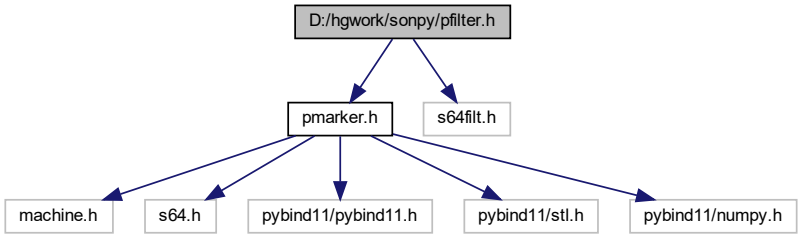
Implements the MarkerFilter class.

6.3 D:/hgwork/sonpy/pfilter.h File Reference

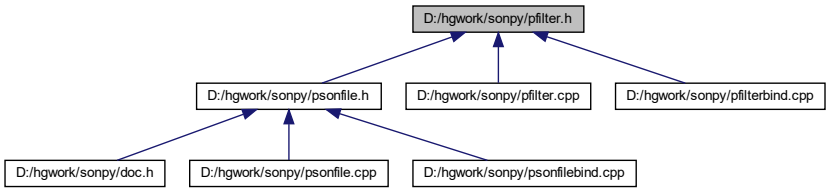
Declares the MarkerFilter class.

```
#include "pmarker.h"
#include "s64filt.h"
```

Include dependency graph for pfilter.h:



This graph shows which files directly or indirectly include this file:



Classes

- class MarkerFilter

  A class for filtering markers.

Enumerations

- enum FilterMode { FilterMode::All = ceds64::CSFilter::eMode::eM_and, FilterMode::First = ceds64::CS↩ Filter::eMode::eM_or }

  How a filter determines which layer to sort with.
- enum FilterSet { FilterSet::Clear = ceds64::CSFilter::eSet::eS_clr, FilterSet::Invert = ceds64::CSFilter::eSet↩ ::eS_inv, FilterSet::Set = ceds64::CSFilter::eSet::eS_set }

  Used to query or set the item states in filter layers.
- enum FilterState { FilterState::All = ceds64::CSFilter::eActive::eA_all, FilterState::None = ceds64::CSFilter↩ ::eActive::eA_none, FilterState::Some = ceds64::CSFilter::eActive::eA_some, FilterState::Unset = ceds64↩ ::CSFilter::eActive::eA_unset }

  Used internally to save some time on calculations, or to get a general description of what the filter does.

6.3.1 Detailed Description

Declares the MarkerFilter class.

6.3.2 Enumeration Type Documentation

6.3.2.1 FilterMode enum FilterMode [strong]

How a filter determines which layer to sort with.

Enumerator

All	Compare all four marker codes against their respective filter layer.
First	Only check the first filter layer for the first marker code.

6.3.2.2 FilterSet enum FilterSet [strong]

Used to query or set the item states in filter layers.

Enumerator

Clear	Markers with this code will not pass the filter.
Invert	Only used with SetItem(). Inverts the current state, whichever it may be.
Set	Markers with this code will pass the filter.

6.3.2.3 FilterState enum FilterState [strong]

Used internally to save some time on calculations, or to get a general description of what the filter does.

Enumerator

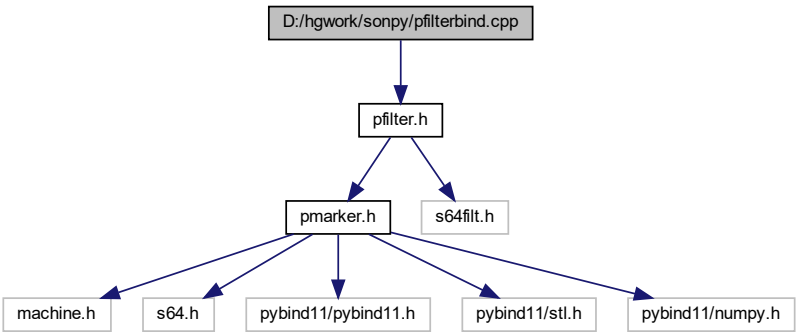
All	This filter will pass all markers.
None	this filter will pass no markers
Some	this filter will pass only some markers
Unset	the state has not yet been determined (you should never see this, as if you call GetState() this attribute will be calculated)

6.4 D:/hgwork/sonpy/pfilterbind.cpp File Reference

Binds the MarkerFilter class.

```
#include "pfilter.h"
```

Include dependency graph for pfilterbind.cpp:



6.4.1 Detailed Description

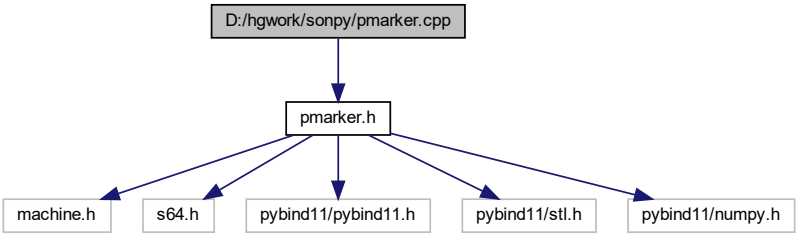
Binds the MarkerFilter class.

6.5 D:/hgwork/sonpy/pmarker.cpp File Reference

Implements DigMark.

```
#include "pmarker.h"
```

Include dependency graph for pmarker.cpp:



6.5.1 Detailed Description

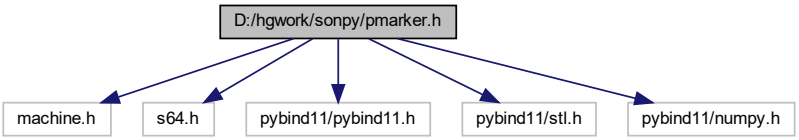
Implements DigMark.

6.6 D:/hgwork/sonpy/pmarker.h File Reference

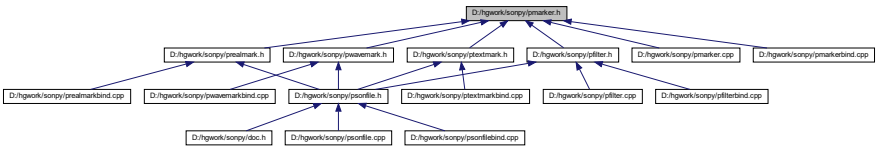
Declares the DigMark struct.

```
#include "machine.h"
#include "s64.h"
#include <pybind11/pybind11.h>
#include <pybind11/stl.h>
#include <pybind11/numpy.h>
```

Include dependency graph for pmarker.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct DigMark

  A simple digital marker.

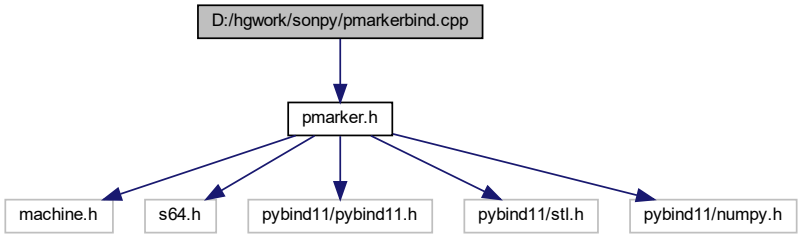
6.6.1 Detailed Description

Declares the DigMark struct.

6.7 D:/hgwork/sonpy/pmarkerbind.cpp File Reference

Binds the DigMark struct.

```
#include "pmarker.h"
```
Include dependency graph for pmarkerbind.cpp:



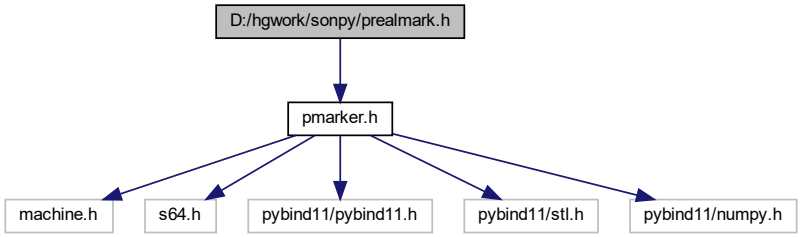
6.7.1 Detailed Description

Binds the DigMark struct.

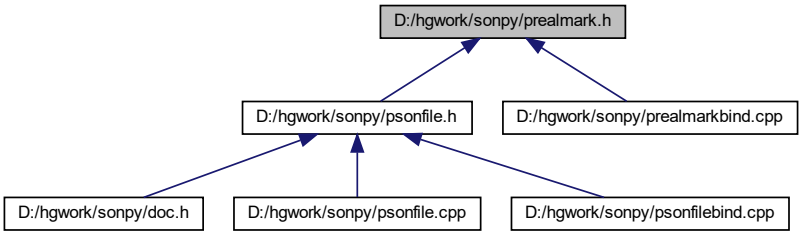
6.8 D:/hgwork/sonpy/prealmark.h File Reference

Declares the RealMarker struct.

```
#include "pmarker.h"
```
Include dependency graph for prealmark.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct RealMarker

6.8.1 Detailed Description

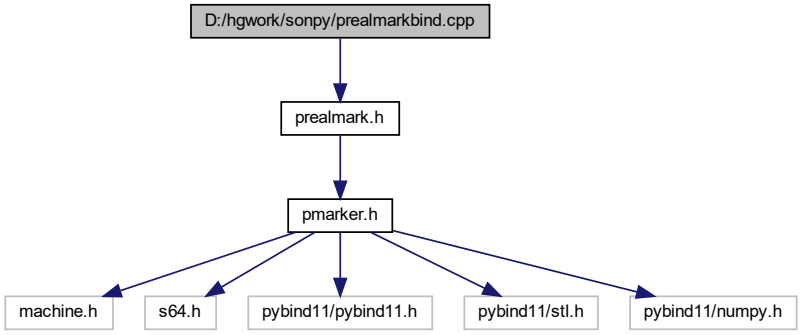
Declares the RealMarker struct.

6.9 D:/hgwork/sonpy/prealmarkbind.cpp File Reference

Binds the RealMarker struct.

```
#include "prealmark.h"
```

Include dependency graph for prealmarkbind.cpp:



6.9.1 Detailed Description

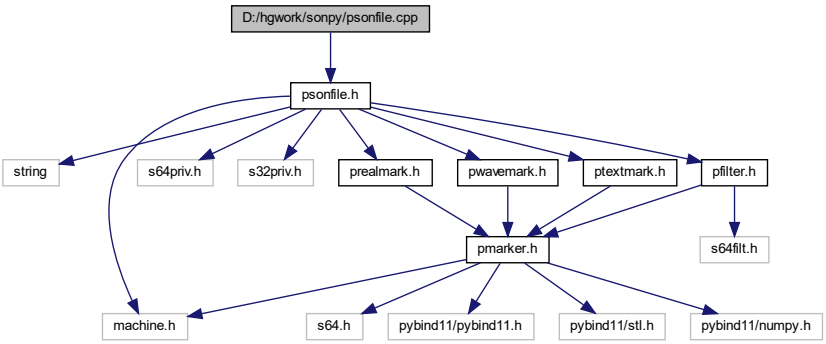
Binds the RealMarker struct.

6.10 D:/hgwork/sonpy/psonfile.cpp File Reference

Implements the SonFile class.

```
#include "psonfile.h"
```

Include dependency graph for psonfile.cpp:



Functions

- std::string GetErrorString (int iErr)

  Converts an error code to a string describing the error.
- int64_t MaxTime64 ()

  Gets the maximum time allowed in a 64-bit SON file.

6.10.1 Detailed Description

Implements the SonFile class.

6.10.2 Function Documentation

6.10.2.1 GetErrorString() `std::string GetErrorString ( int iErr )`

Converts an error code to a string describing the error.

Parameters

iErr	Error code corresponding to a SonError

Returns

The error code described as a string

6.10.2.2 MaxTime64() `int64_t MaxTime64 ( )`

Gets the maximum time allowed in a 64-bit SON file.

Returns

The value of the largest possible number of (usable) ticks in a 64-bit SON file.

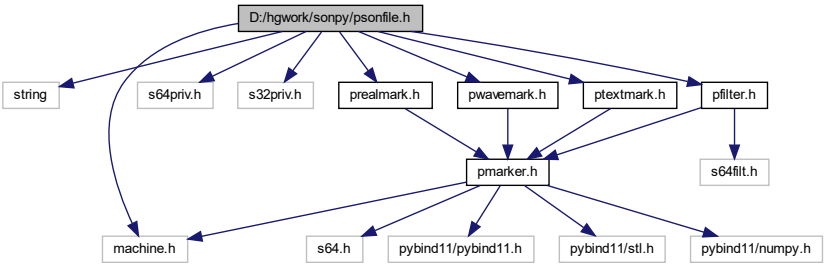
6.11 D:/hgwork/sonpy/psonfile.h File Reference

Declares the SonFile class.

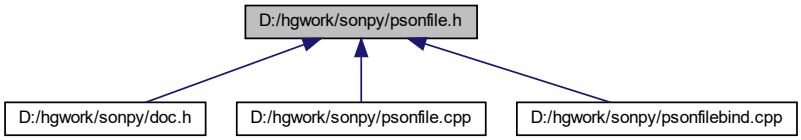
```
#include <string>
#include "machine.h"
#include "s64priv.h"
#include "s32priv.h"
#include "pfilter.h"
```

```
#include "prealmark.h"
#include "pwavemark.h"
#include "ptextmark.h"
```

Include dependency graph for psonfile.h:



This graph shows which files directly or indirectly include this file:



Classes

- class SonFile

  Wraps either a 64- or 32-bit File and passes function calls to it in a tidy way.

Enumerations

- enum OpenFlags { OpenFlags::None = ceds64::eOpenFlags::eOF_none, OpenFlags::Shared = ceds64::e↩ OpenFlags::eOF_shared, OpenFlags::Test = ceds64::eOpenFlags::eOF_test }

  Options for file opening.
- enum SonError {
  SonError::Son_OK = ceds64::S64_ERROR::S64_OK, SonError::No_File = ceds64::S64_ERROR::NO_FILE,
  SonError::No_Block = ceds64::S64_ERROR::NO_BLOCK, SonError::Call_Again = ceds64::S64_ERROR↩ ::CALL_AGAIN,
  SonError::No_Access = ceds64::S64_ERROR::NO_ACCESS, SonError::No_Memory = ceds64::S64_ER↩ ROR::NO_MEMORY, SonError::No_Channel = ceds64::S64_ERROR::NO_CHANNEL, SonError::Channel_Used = ceds64::S64_ERROR::CHANNEL_USED,
  SonError::Channel_Type = ceds64::S64_ERROR::CHANNEL_TYPE, SonError::Past_EOF = ceds64::S64↩ _ERROR::PAST_EOF, SonError::Wrong_File = ceds64::S64_ERROR::WRONG_FILE, SonError::No_Extra = ceds64::S64_ERROR::NO_EXTRA,
  SonError::Bad_Read = ceds64::S64_ERROR::BAD_READ, SonError::Bad_Write = ceds64::S64_ERROR↩ ::BAD_WRITE, SonError::Corrupt_File = ceds64::S64_ERROR::CORRUPT_FILE, SonError::Past_SOF = ceds64::S64_ERROR::PAST_SOF,
  SonError::Read_Only = ceds64::S64_ERROR::READ_ONLY, SonError::Bad_Param = ceds64::S64_ERR↩ OR::BAD_PARAM, SonError::Over_Write = ceds64::S64_ERROR::OVER_WRITE, SonError::More_Data = ceds64::S64_ERROR::MORE_DATA }

Possible errors that may be returned by Son64 library functions.

- enum DataType { DataType::Off = ceds64::TDataKind::ChanOff, DataType::Adc = ceds64::TDataKind::Adc, DataType::EventFall = ceds64::TDataKind::EventFall, DataType::EventRise = ceds64::TDataKind::EventRise, DataType::EventBoth = ceds64::TDataKind::EventBoth, DataType::Marker = ceds64::TDataKind::Marker, DataType::AdcMark = ceds64::TDataKind::AdcMark, DataType::RealMark = ceds64::TDataKind::RealMark, DataType::TextMark = ceds64::TDataKind::TextMark, DataType::RealWave = ceds64::TDataKind::RealWave }

  The different possible data types a channel can be set to hold.
- enum CommitFlags { CommitFlags::None = 0, CommitFlags::FlushSystem = ceds64::eCommitFlags::eCF_flushSys, CommitFlags::HeaderOnly = ceds64::eCommitFlags::eCF_headerOnly, CommitFlags::DeleteBuffer = ceds64::eCommitFlags::eCF_delBuffer }

  Flags for commiting data to disk (see SonFile::Commit())

Functions

- std::string GetErrorString (int iErr)

  Converts an error code to a string describing the error.
- int64_t MaxTime64 ()

  Gets the maximum time allowed in a 64-bit SON file.

6.11.1 Detailed Description

Declares the SonFile class.

6.11.2 Enumeration Type Documentation

6.11.2.1 CommitFlags `enum CommitFlags [strong]`

Flags for commiting data to disk (see SonFile::Commit())

Enumerator

None	No extra behaviour
FlushSystem	After writing data, tell system to ensure it is physically on the disk. This can be VERY slow (several seconds). Only use this if you are certain you can tolerate the disk system becoming inaccessible for several seconds. You might use this just before closing a newly created file to be certain that it is safely saved. Beware that on Windows systems this flag has been observed to cause the system to become unresponsive for several seconds while all dirty data buffered on the disk is written. We suspect that this is because modern disks may implement very large data buffers. They know which blocks in the buffers are dirty (need writing), but may not know which file each block belongs to. When we tell the OS to commit a file with FlushSystem, the system tells the disk to write all dirty blocks, and this can be a lot of data. Ironically, older disks that rely on the OS to handle all buffering work better for a commit as the OS can write only dirty blocks belonging to the data file.
HeaderOnly	Only commit the file header, not buffered channel data. You might use this if you added or deleted a channel or changed any file information and wanted to be certain that the file header was committed with the change. The file header also holds all text strings used in the file (even channel comments, titles and units)
DeleteBuffer	Kill off channel buffering after committing data. This will set any circular buffer set for the channel to have 0 size. If you use this flag with HeaderOnly you will lose any data that was in the circular buffer.

6.11.2.2 DataType `enum DataType [strong]`

The different possible data types a channel can be set to hold.

There are two wave type, three event types and four marker types. Any marker type that is not the standard marker is also known as an extended marker type.

There is a hierarchy to the non-wave types. At the bottom are the event types. All are essentially the same thing for the purposes of data storage, it is only in their interpretation that they differ. These are built upon by markers, which in turn are build upon by any of the extended markers. For details on how this works, see the description of each type.

Wave type channels can have data overwritten, although this is not intended behaviour. It is expected that the only use of this would be to fix holes in corrupt data. In normal use, all data for a wave type channel will be aligned at times that are exact multiples of the channel divide after the first data point. We do not prevent you writing data where the alignment is not the same after a gap, but Spike2 will experience subtle problems if you do this.

All other (event/marker based) channels must be time ordered (monotonically increasing) or the file will be corrupted. You cannot write events or markers to a point in time earlier than any already existing items.

Enumerator

Off	This channel is not used and has no data associated with it.
Adc	Sometimes known as WaveForm data. This consists of time-continuous data stored as short integers (i.e. 16-bits long). This is the most fundamental type of channel to Spike2. Each Adc channel contains a scale and an offset, so that data in the 16-bit range (-32768 to 32767) can be transformed onto any required range according to y=mx+c. For more details about how the scale and offset are used, see SetChannelScale()
EventFall	Essentially a list of points at which a signal has passed from high to low. Each item constists of a single 64-bit integer which contains the time value, in ticks, at which the event occured.
EventRise	As for EventFall, but corresponds to rising, rather than falling, triggers.
EventBoth	As with other event channels, a list of times, this time, when a signal passes either direction, high to low or low to high.
Marker	This type builds on events by having 4 8-bit codes in addition to the 64-bit time stamp. There are also 32 bits of reserved space after these codes, so the whole structure is 16 bytes long.
AdcMark	Also called WaveMark data, this extended marker type augments the standard marker with a series of short integers (16-bit). If you set up a channel of this type with multiple columns, each column represents a different 'trace'. Each trace is essentially a wave fragment that is associated with its marker. If you associate a timebase with the channel, it can be interpreted in context of the timebase of the entire file, and Spike2 can plot the data as isolated snippets of Adc data. If you use multiple traces, the data is stored interleaved.
RealMark	Similar to AdcMark, this exteneded marker augments the standard marker with a series of 4 byte floating point data. Unlike AdcMark however, only a single column of data is supported, as this type is not used to draw multiple waves, rather to store multiple single measurements.
TextMark	This extended marker augments the standard marker with a zero-terminated series of characters. You can store a string of any length in each, but Spike2 has a limit of 79 characters, so anything longer than this will be truncated if you open the file in Spike2.
RealWave	Similar to Adc data, this consists of a contiguous stream of 4 byte float data. There is a scale and offset associated with these channels, but it is most often entirely redundant. For more details about how the scale and offset are used, see SetChannelScale().

6.11.2.3 OpenFlags enum OpenFlags [strong]

Options for file opening.

The file open constructor uses None by default, so you never have to worry about these.

Enumerator

None	Business as usual.
Shared	Currently redundant.
Test	Open file even if header can't be verified.

6.11.2.4 SonError enum SonError [strong]

Possible errors that may be returned by Son64 library functions.

To get descriptions of one of these error from within Pyhon, see GetErrorString().

Enumerator

Son_OK	No error.
No_File	This object does not own a file handle or any resources.
No_Block	Failed to allocate a disk block.
Call_Again	Long operation, call again (this should only be visible internally!)
No_Access	No access: bad operation or file in use.
No_Memory	Out of memory reading 32-bit file.
No_Channel	Channel doesn't exist.
Channel_Used	Channel already in use.
Channel_Type	Channel has wrong type.
Past_EOF	Tried to access past the end of the file.
Wrong_File	Tried to open wrong file type.
No_Extra	Request is outside the extra data region.
Bad_Read	Read error.
Bad_Write	Write error.
Corrupt_File	File is bad or tired to write corrupt data.
Past_SOF	Tried to access before the start of the file.
Read_Only	Tried to write to a read only file.
Bad_Param	A parameter is bad (check type, dimensions and sizes)
Over_Write	Tried to over-write data when not allowed.
More_Data	The file is bigger than the header says; it may not have closed properly.

6.11.3 Function Documentation

6.11.3.1 GetErrorString() `std::string GetErrorString ( int iErr )`

Converts an error code to a string describing the error.

Parameters

iErr	Error code corresponding to a SonError

Returns

The error code described as a string

6.11.3.2 MaxTime64() `int64_t MaxTime64 ( )`

Gets the maximum time allowed in a 64-bit SON file.

Returns

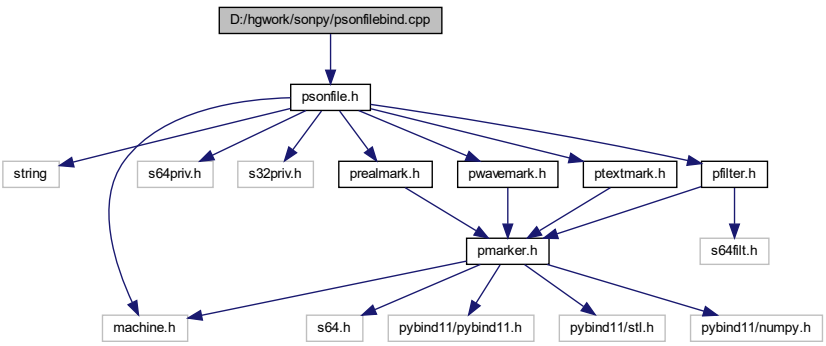
The value of the largest possible number of (usable) ticks in a 64-bit SON file.

6.12 D:/hgwork/sonpy/psonfilebind.cpp File Reference

Binds the SonFile class.

```
#include "psonfile.h"
```

Include dependency graph for psonfilebind.cpp:



6.12.1 Detailed Description

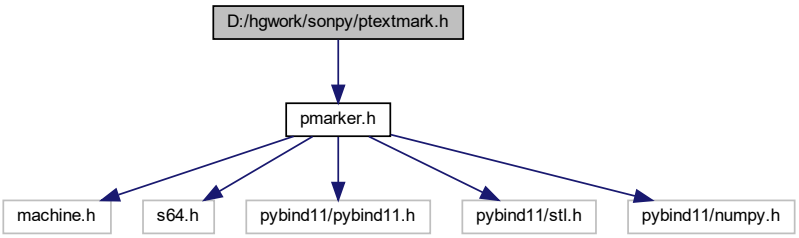
Binds the SonFile class.

6.13 D:/hgwork/sonpy/ptextmark.h File Reference

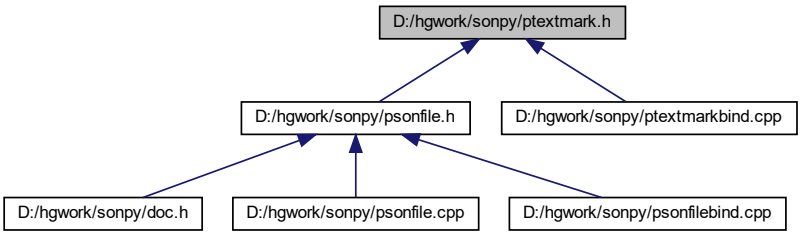
Declares the TextMarker struct.

```
#include "pmarker.h"
```

Include dependency graph for ptextmark.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct TextMarker

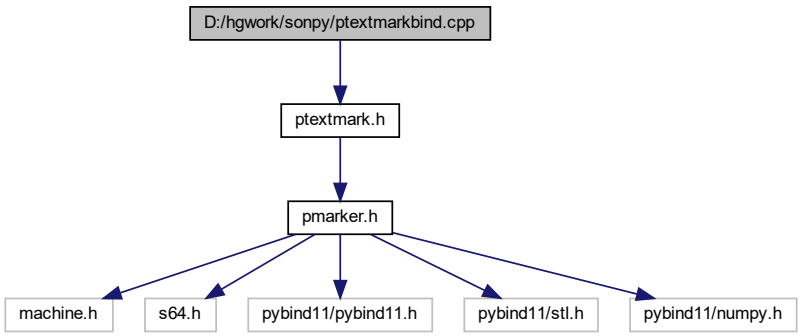
6.13.1 Detailed Description

Declares the TextMarker struct.

6.14 D:/hgwork/sonpy/ptextmarkbind.cpp File Reference

Binds the TextMarker struct.

`#include "ptextmark.h"`
Include dependency graph for ptextmarkbind.cpp:



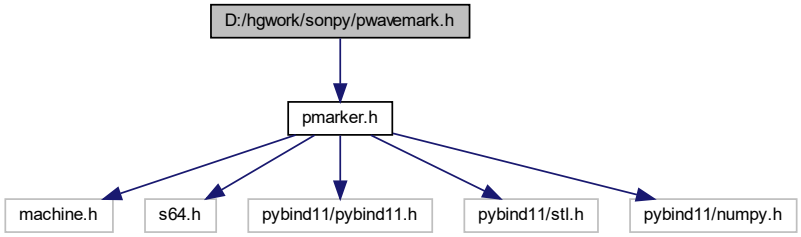
6.14.1 Detailed Description

Binds the TextMarker struct.

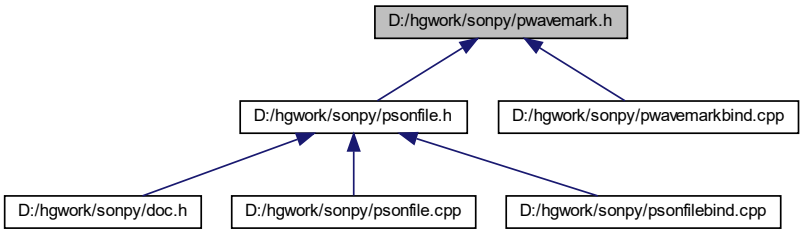
6.15 D:/hgwork/sonpy/pwavemark.h File Reference

Declares the WaveMarker struct.

`#include "pmarker.h"`
Include dependency graph for pwavemark.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct WaveMarker

6.15.1 Detailed Description

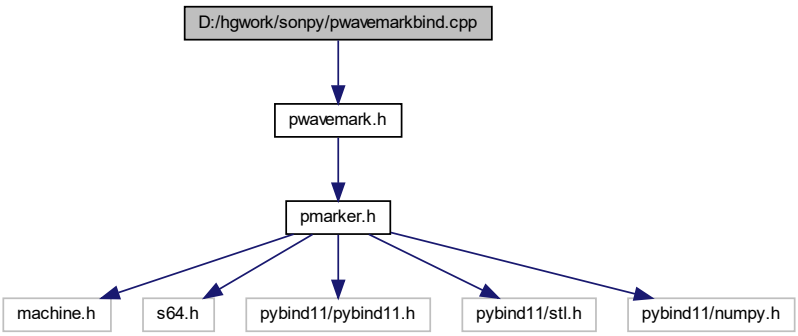
Declares the WaveMarker struct.

6.16 D:/hgwork/sonpy/pwavemarkbind.cpp File Reference

Binds the WaveMarker struct.

```
#include "pwavemark.h"
```

Include dependency graph for pwavemarkbind.cpp:



6.16.1 Detailed Description

Binds the WaveMarker struct.

Index