mdfreader Documentation

Release 1.1

Aymeric Rateau

May 11, 2015

Contents:

MDFREADER MODULE DOCUMENTATION

Measured Data Format file reader main module

1.1 Platform and python version

With Unix and Windows for python 2.6+ and 3.2+

> **Author** Aymeric Rateau

Created on Sun Oct 10 12:57:28 2010

1.2 Dependencies

- Python >2.6, >3.2 <http://www.python.org>
- Numpy >1.6 <http://numpy.scipy.org>
- Sympy to convert channels with formula
- bitarray for not byte aligned data parsing
- Matplotlib >1.0 <http://matplotlib.sourceforge.net>
- NetCDF
- h5py for the HDF5 export
- xlwt for the excel export (not existing for python3)
- openpyxl for the excel 2007 export
- scipy for the Matlab file conversion
- zlib to uncompress data block if needed

1.3 Attributes

PythonVersion [float] Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

1.4 mdfreader module

mdfreader.**convertMatlabName**(*channel*)

> Removes non allowed characters for a Matlab variable name

> > **Parameters channel** : string
> >
> > > channel name
> >
> > **Returns** string
> >
> > > channel name compatible for Matlab

class mdfreader.**mdf**(*fileName=None, channelList=None, convertAfterRead=True, filterChannel-Names=False*)

> Bases: `mdf3reader.mdf3`, `mdf4reader.mdf4`

> mdf class

> ### Notes

> mdf class is a nested dict Channel name is the primary dict key of mdf class At a higher level, each channel includes the following keys :

> > • 'data' : containing vector of data (numpy)
> >
> > • 'unit' : unit (string)
> >
> > • 'master' : master channel of channel (time, crank angle, etc.)
> >
> > • 'description' : Description of channel
> >
> > • **'conversion': mdfinfo nested dict for CCBlock.** Exist if channel not converted, used to convert with getChannelData method

> ### Examples

```
>>> import mdfreader
>>> yop=mdfreader.mdf('NameOfFile')
>>> yop.keys() # list channels names
>>> yop.masterChannelList() # list channels grouped by raster or master channel
>>> yop.plot('channelName') or yop.plot({'channel1','channel2'})
>>> yop.resample(0.1) or yop.resample(channelName='master3')
>>> yop.exportoCSV(sampling=0.01)
>>> yop.exportNetCDF()
>>> yop.exporttoHDF5()
>>> yop.exporttoMatlab()
>>> yop.exporttoExcel()
>>> yop.exporttoXlsx()
>>> yop.convertToPandas() # converts data groups into pandas dataframes
>>> yop.keepChannels({'channel1','channel2','channel3'}) # drops all the channels except the one
>>> yop.getChannelData('channelName') # returns channel numpy array
```

Attributes

fileName	(str) file name
Version-Number	(int) mdf file version number
master-Channel-List	(dict) Represents data structure: a key per master channel with corresponding value containing a list of channels One key or master channel represents then a data group having same sampling interval.
multiProc	(bool) Flag to request channel conversion multi processed for performance improvement. One thread per data group.
author	(str)
organisa-tion	(str)
project	(str)
subject	(str)
comment	(str)
time	(str)
date	(str)

Methods

read(fileName = None, multiProc = False, channelList=None, convertAfterRead=True, filterChannelNames=False)	reads mdf file version 3.x and 4.x
write(fileName=None)	writes simple mdf 3.3 file
getChannelData(channelName)	returns channel numpy array
convertAllChannel()	converts all channel data according to CCBlock information
getChannelUnit(channelName)	returns channel unit
plot(channels)	Plot channels with Matplotlib
resample(samplingTime = 0.1, masterChannel=None)	Resamples all data groups
exportToCSV(filename = None, sampling = 0.1)	Exports mdf data into CSV file
exportToNetCDF(filename = None, sampling = None)	Exports mdf data into netcdf file
exportToHDF5(filename = None, sampling = None)	Exports mdf class data structure into hdf5 file
exportToMatlab(filename = None)	Exports mdf class data structure into Matlab file
exportToExcel(filename = None)	Exports mdf data into excel 95 to 2003 file
exportToXlsx(filename=None)	Exports mdf data into excel 2007 and 2010 file
convertToPandas(sampling=None)	converts mdf data structure into pandas dataframe(s)
keepChannels(channelList)	keeps only list of channels and removes the other channels
mergeMdf(mdfClass):	Merges data of 2 mdf classes
copy()	copy a mdf class

allPlot()

convertAllChannel()
> Converts all channels from raw data to converted data according to CCBlock information Converted data will take more memory.

convertToPandas (*sampling=None*)
   converts mdf data structure into pandas dataframe(s)

> **Parameters sampling** : float, optional
>
>> resampling interval

> ### Notes

> One pandas dataframe is converted per data group Not adapted yet for mdf4 as it considers only time master channels

copy ( )
   copy a mdf class

exportToCSV (*filename=None*, *sampling=0.1*)
   Exports mdf data into CSV file

> **Parameters filename** : str, optional
>
>> file name. If no name defined, it will use original mdf name and path
>
> **sampling** : float, optional
>
>> sampling interval. By default, sampling is 0.1sec but can be changed

> ### Notes

> Data saved in CSV fille be automatically resampled as it is difficult to save in this format data not sharing same master channel Warning: this can be slow for big data, CSV is text format after all

exportToExcel (*filename=None*)
   Exports mdf data into excel 95 to 2003 file

> **Parameters filename** : str, optional
>
>> file name. If no name defined, it will use original mdf name and path

> ### Notes

> xlwt is not fast for even for small files, consider other binary formats like HDF5 or Matlab If there are more than 256 channels, data will be saved over different worksheets Also Excel 203 is becoming rare these days

exportToHDF5 (*filename=None*, *sampling=None*)
   Exports mdf class data structure into hdf5 file

> **Parameters filename** : str, optional
>
>> file name. If no name defined, it will use original mdf name and path
>
> **sampling** : float, optional
>
>> sampling interval.

Notes

The maximum attributes will be stored Data structure will be similar has it is in masterChannelList attribute

exportToMatlab (*filename=None*)
Export mdf data into Matlab file format 5, tentatively compressed

> **Parameters filename** : str, optional
>
> > file name. If no name defined, it will use original mdf name and path

Notes

This method will dump all data into Matlab file but you will loose below information: - unit and descriptions of channel - data structure, what is corresponding master channel to a channel. Channels might have then different lengths

exportToNetCDF (*filename=None*, *sampling=None*)
Exports mdf data into netcdf file

> **Parameters filename** : str, optional
>
> > file name. If no name defined, it will use original mdf name and path
>
> > **sampling** : float, optional
>
> > sampling interval.

exportToXlsx (*filename=None*)
Exports mdf data into excel 2007 and 2010 file

> **Parameters filename** : str, optional
>
> > file name. If no name defined, it will use original mdf name and path

Notes

It is recommended to export resampled data for performances

getChannelData (*channelName*)
Return channel numpy array

> **Parameters channelName** : str
>
> > channel name

Notes

This method is the safest to get channel data as numpy array from 'data' dict key might contain raw data

getChannelUnit (*channelName*)
Returns channel unit string Implemented for a future integration of pint

> **Parameters channelName** : str
>
> > channel name
>
> **Returns** str
>
> > unit string description

keepChannels (*channelList*)

  keeps only list of channels and removes the other channels

> **Parameters channelList** : list of str
>
>> list of channel names

mergeMdf (*mdfClass*)

  Merges data of 2 mdf classes

> **Parameters mdfClass** : mdf
>
>> mdf class instance to be merge with self

> ### Notes

> both classes must have been resampled, otherwise, impossible to know master channel to match create union of both channel lists and fill with Nan for unknown sections in channels

plot (*channels*)

  Plot channels with Matplotlib

> **Parameters channels** : str or list of str
>
>> channel name or list of channel names

> ### Notes

> Channel description and unit will be tentatively displayed with axis labels

read (*fileName=None*, *multiProc=False*, *channelList=None*, *convertAfterRead=True*, *filterChannelNames=False*)

  reads mdf file version 3.x and 4.x

> **Parameters fileName** : str, optional
>
>> file name
>
> **multiProc** : bool
>
>> flag to activate multiprocessing of channel data conversion
>
> **channelList** : list of str, optional
>
>> list of channel names to be read If you use channelList, reading might be much slower but it will save you memory. Can be used to read big files
>
> **convertAfterRead** : bool, optional
>
>> flag to convert channel after read, True by default If you use convertAfterRead by setting it to false, all data from channels will be kept raw, no conversion applied. If many float are stored in file, you can gain from 3 to 4 times memory footprint To calculate value from channel, you can then use method .getChannelData()
>
> **filterChannelNames** : bool, optional
>
>> flag to filter long channel names from its module names separated by '.'

Notes

If you keep convertAfterRead to true, you can set attribute mdf.multiProc to activate channel conversion in multiprocessing. Gain in reading time can be around 30% if file is big and using a lot of float channels

resample(*samplingTime=0.1*, *masterChannel=None*)

Resamples all data groups into one data group having defined sampling interval or sharing same master channel

> **Parameters samplingTime** : float
>
> > resampling interval
>
> ****or****
>
> **masterChannel** : str
>
> > master channel name used for all channels

Notes

1. resampling is relatively safe for mdf3 as it contains only time series. However, mdf4 can contain also distance, angle, etc. It might make not sense to apply one resampling to several data groups that do not share same kind of master channel (like time resampling to distance or angle data groups) If several kind of data groups are used, you should better use pandas to resample

2. resampling will convert all your channels so be careful for big files and memory consumption

write(*fileName=None*)

Writes simple mdf 3.3 file

> **Parameters fileName** : str, optional
>
> > Name of file If file name is not input, written file name will be the one read with appended '_new' string before extension

Notes

All channels will be converted, so size might be bigger than original file

class mdfreader.**mdfinfo**(*fileName=None*, *filterChannelNames=False*)

Bases: dict

MDFINFO is a class gathering information from block headers in a MDF (Measure Data Format) file

Structure: nested dicts. Primary key is Block type, then data group, channel group and channel number. Examples of dicts:

• mdfinfo['HDBlock'] header block

• mdfinfo['DGBlock'][dataGroup] Data Group block

• mdfinfo['CGBlock'][dataGroup][channelGroup] Channel Group block

• mdfinfo['CNBlock'][dataGroup][channelGroup][channel] Channel block including text blocks for comment and identifier

• mdfinfo['CCBlock'][dataGroup][channelGroup][channel] Channel conversion information

Examples

```python
>>> import mdfreader
>>> FILENAME='toto.dat'
>>> yop=mdfreader.mdfinfo(FILENAME)
or if you are just interested to have only list of channels
>>> yop=mdfreader.mdfinfo() # creates new instance f mdfinfo class
>>> yop=mdfreader.listChannels(FILENAME) # returns a simple list of channel names
```

Attributes

fileName	(str) file name
mdfversion	(int) mdf file version number

Methods

readinfo(fileName = None, filterChannelNames=False)	Reads MDF file and extracts its complete structure
listChannels(fileName = None)	Read MDF file blocks and returns a list of contained channels

listChannels (*fileName=None*)
    Read MDF file blocks and returns a list of contained channels

> **Parameters fileName** : string

>> file name

> **Returns nameList** : list of string

>> list of channel names

readinfo (*fileName=None*, *filterChannelNames=False*)
    Reads MDF file and extracts its complete structure

> **Parameters fileName** : str, optional

>> file name. If not input, uses fileName attribute

>> **filterChannelNames** : bool, optional

>> flag to filter long channel names including module names separated by a '.'

MDF3READER MODULE DOCUMENTATION

Measured Data Format file reader module for version 3.x

2.1 Platform and python version

With Unix and Windows for python 2.6+ and 3.2+

Author Aymeric Rateau

Created on Sun Oct 10 12:57:28 2010

2.2 Dependencies

- Python >2.6, >3.2 <http://www.python.org>
- Numpy >1.6 <http://numpy.scipy.org>
- Sympy to convert channels with formula

2.3 Attributes

PythonVersion [float] Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

2.4 mdf3reader module

class `mdf3reader.`**`DATA`**(*fid*, *pointer*)
  Bases: `dict`

  DATA class is organizing record classes itself made of recordchannel. This class inherits from dict. Keys are corresponding to channel group recordID A DATA class corresponds to a data block, a dict of record classes (one per channel group) Each record class contains a list of recordchannel class representing the structure of channel record.

Attributes

fid	(io.open) file identifier
pointerToData	(int) position of Data block in mdf file

Methods

addRecord(record)	Adds a new record in DATA class dict
read(channelList, zip=None)	Reads data block
loadSorted(record, zip=None, nameList=None)	Reads sorted data block from record definition
load(nameList=None)	Reads unsorted data block, not yet implemented

addRecord(*record*)
    Adds a new record in DATA class dict

>   **Parameters record class**

>   channel group definition listing record channel classes

load(*nameList=None*)
    not yet implemented

loadSorted(*record*, *zip=None*, *nameList=None*)
    Reads sorted data block from record definition

>   **Parameters record class**

>   channel group definition listing record channel classes

>   **zip** : bool, optional

>   flag to track if data block is compressed

>   **channelList** : list of str, optional

>   list of channel names

>   **Returns** numpy recarray of data

read(*channelList*, *zip=None*)
    Reads data block

>   **Parameters channelList** : list of str, optional

>   list of channel names

>   **zip** : bool, optional

>   flag to track if data block is compressed

mdf3reader.**arrayformat3**(*signalDataType*, *numberOfBits*)
    function returning numpy style string from channel data type and number of bits Parameters ——————-
    signalDataType : int

>   channel data type according to specification

>   **numberOfBits** [int] number of bits taken by channel data in a record

>   **Returns dataType** : str

>   numpy dtype format used by numpy.core.records to read channel raw data

mdf3reader.**datatypeformat3**(*signalDataType*, *numberOfBits*)
    function returning C format string from channel data type and number of bits

>   **Parameters signalDataType** : int

>   channel data type according to specification

>   **numberOfBits** : int

number of bits taken by channel data in a record

Returns dataType : str

C format used by fread to read channel raw data

`mdf3reader.`**`expConv`**(*data*, *conv*)

apply exponential conversion to data

Parameters data : numpy 1D array

raw data to be converted to physical value

conv : mdfinfo3.info3 conversion block ('CCBlock') dict

Returns converted data to physical value

`mdf3reader.`**`formulaConv`**(*data*, *conv*)

apply formula conversion to data

Parameters data : numpy 1D array

raw data to be converted to physical value

conv : mdfinfo3.info3 conversion block ('CCBlock') dict

Returns converted data to physical value

Notes

Requires sympy module

`mdf3reader.`**`linearConv`**(*data*, *conv*)

apply linear conversion to data

Parameters data : numpy 1D array

raw data to be converted to physical value

conv : mdfinfo3.info3 conversion block ('CCBlock') dict

Returns converted data to physical value

`mdf3reader.`**`logConv`**(*data*, *conv*)

apply logarithmic conversion to data

Parameters data : numpy 1D array

raw data to be converted to physical value

conv : mdfinfo3.info3 conversion block ('CCBlock') dict

Returns converted data to physical value

class `mdf3reader.`**`mdf3`**(*fileName=None*, *info=None*, *multiProc=False*, *channelList=None*, *convertAfter-Read=True*, *filterChannelNames=False*)

Bases: `dict`

mdf file version 3.0 to 3.3 class

Attributes

fileName	(str) file name
Version-Number	(int) mdf file version number
master-Channel-List	(dict) Represents data structure: a key per master channel with corresponding value containing a list of channels One key or master channel represents then a data group having same sampling interval.
multiProc	(bool) Flag to request channel conversion multi processed for performance improvement. One thread per data group.
con-vertAfter-Read	(bool) flag to convert raw data to physical just after read
filterChan-nelNames	(bool) flag to filter long channel names from its module names separated by '.'
author	(str)
organisa-tion	(str)
project	(str)
subject	(str)
comment	(str)
time	(str)
date	(str)

Methods

read3(fileName=None, info=None, multiProc=False, channelList=None, convertAfterRead=True)	Reads mdf 3.x file data and stores it in dict
getChannelData3(channelName)	Returns channel numpy array
convertChannel3(channelName)	converts specific channel from raw to physical data according to CCBlock information
convertAllChannel3()	Converts all channels from raw data to converted data according to CCBlock information
write3(fileName=None)	Writes simple mdf 3.3 file

convert3(*channelName*)

converts specific channel from raw to physical data according to CCBlock information

>**Parameters channelName** : str

>>Name of channel

>**Returns** numpy array

>>returns numpy array converted to physical values according to conversion type

convertAllChannel3()

Converts all channels from raw data to converted data according to CCBlock information Converted data will take more memory.

convertChannel3(*channelName*)

converts specific channel from raw to physical data according to CCBlock information

>**Parameters channelName** : str

>>Name of channel

getChannelData3(*channelName*)
  Returns channel numpy array

  Parameters **channelName** : str

  channel name

  Notes

  This method is the safest to get channel data as numpy array from 'data' dict key might contain raw data

read3(*fileName=None*, *info=None*, *multiProc=False*, *channelList=None*, *convertAfterRead=True*)
  Reads mdf 3.x file data and stores it in dict

  Parameters **fileName** : str, optional

  file name

  info : mdfinfo3.info3 class

  info3 class containing all MDF Blocks

  multiProc : bool

  flag to activate multiprocessing of channel data conversion

  channelList : list of str, optional

  list of channel names to be read If you use channelList, reading might be much slower but it will save you memory. Can be used to read big files

  convertAfterRead : bool, optional

  flag to convert channel after read, True by default If you use convertAfterRead by setting it to false, all data from channels will be kept raw, no conversion applied. If many float are stored in file, you can gain from 3 to 4 times memory footprint To calculate value from channel, you can then use method .getChannelData()

write3(*fileName=None*)
  Writes simple mdf 3.3 file

  Parameters **fileName** : str, optional

  Name of file If file name is not input, written file name will be the one read with appended '_new' string before extension

  Notes

  All channels will be converted to physical data, so size might be bigger than original file

mdf3reader.**polyConv**(*data*, *conv*)
  apply polynomial conversion to data

  Parameters **data** : numpy 1D array

  raw data to be converted to physical value

  conv : mdfinfo3.info3 conversion block ('CCBlock') dict

  Returns converted data to physical value

mdf3reader.**processDataBlocks**(*Q*, *buf*, *info*, *dataGroup*, *channelList*, *multiProc*)
  Put raw data from buf to a dict L and processes nested nBit channels

---

> **Parameters** **Q** : multiprocessing.Queue, optional
>
>> Queue for multiprocessing
>
>> **buf** : DATA class
>
>> contains raw data
>
>> **info** : info class
>
>> contains infomation from MDF Blocks
>
>> **dataGroup** : int
>
>> data group number according to info class
>
>> **channelList** : list of str, optional
>
>> list of channel names to be processed
>
>> **multiProc** : bool
>
>> flag to return Queue or dict
>
> **Returns** **Q** : multiprocessing.Queue
>
>> updates Queue containing L dict
>
>> **L** : dict
>
>> dict of channels

mdf3reader.**rationalConv**(*data*, *conv*)

> apply rational conversion to data
>
>> **Parameters** **data** : numpy 1D array
>
>> raw data to be converted to physical value
>
>> **conv** : mdfinfo3.info3 conversion block ('CCBlock') dict
>
>> **Returns** converted data to physical value

class mdf3reader.**record**(*dataGroup*, *channelGroup*)

> Bases: list
>
> record class lists recordchannel classes, it is representing a channel group

Attributes

recordLength	(int) length of record corresponding of channel group in Byte
numberOfRecords	(int) number of records in data block
recordID	(int) recordID corresponding to channel group
recordIDsize	(int) size of recordID
dataGroup	(int:) data group number
channelGroup	(int) channel group number
numpyDataRecordFormat	(list) list of numpy (dtype) for each channel
dataRecordName	(list) list of channel names used for recarray attribute definition
master	(dict) define name and number of master channel
recordToChannelMatch-ing	(dict) helps to identify nested bits in byte
channelNames	(list) channel names to be stored, useful for low memory consumption but slow

Methods

addChannel(info, channelNumber)	
loadInfo(info)	
readSortedRecord(fid, pointer, channelList=None)	
readUnsortedRecord(buf, channelList=None)	

addChannel(*info*, *channelNumber*)
    add a channel in class

        Parameters info : mdfinfo3.info3 class

            channelNumber : int

                channel number in mdfinfo3.info3 class

loadInfo(*info*)
    gathers records related from info class

        Parameters info : mdfinfo3.info3 class

readSortedRecord(*fid*, *pointer*, *channelList=None*)
    reads record, only one channel group per datagroup Parameters ——————- fid : float

        file identifier

    pointer position in file of data block beginning

    channelList [list of str, optional] list of channel to read

        Returns rec : numpy recarray

            contains a matrix of raw data in a recarray (attributes corresponding to channel name)

Notes

    If channelList is None, read data using numpy.core.records.fromfile that is rather quick. However, in case of large file, you can use channelList to load only interesting channels or only one channel on demand, but be aware it might be much slower.

readUnsortedRecord(*buf*, *channelList=None*)
    Not implemented yet, no reference files available to test it

class mdf3reader.**recordChannel**(*info*, *dataGroup*, *channelGroup*, *channelNumber*, *recordIDsize*)
    recordChannel class gathers all about channel structure in a record

Attributes

name	(str) Name of channel
channelNum-ber	(int) channel number corresponding to mdfinfo3.info3 class
signal-DataType	(int) signal type according to specification
bitCount	(int) number of bits used to store channel record
nBytes	(int) number of bytes (1 byte = 8 bits) taken by channel record
dataFormat	(str) numpy dtype as string
CFormat	(struct class instance) struct instance to convert from C Format
byteOffset	(int) position of channel record in complete record in bytes
bitOffset	(int) bit position of channel value inside byte in case of channel having bit count below 8
RecordFormat	(list of str) dtype format used for numpy.core.records functions ((name,name_title),str_stype)
channelType	(int) channel type
posBeg	(int) start position in number of bit of channel record in complete record
posEnd	(int) end position in number of bit of channel record in complete record

Methods

__init__(info, dataGroup, channelGroup, channelNumber, recordIDsize)	constructor
__str__()	to print class attributes

mdf3reader.**tabConv**(*data*, *conv*)

    apply Tabular conversion to data

        Parameters data : numpy 1D array

            raw data to be converted to physical value

            conv : mdfinfo3.info3 conversion block ('CCBlock') dict

        Returns converted data to physical value

mdf3reader.**tabInterpConv**(*data*, *conv*)

    apply Tabular interpolation conversion to data

        Parameters data : numpy 1D array

            raw data to be converted to physical value

            conv : mdfinfo3.info3 conversion block ('CCBlock') dict

        Returns converted data to physical value

mdf3reader.**textRangeTableConv**(*data*, *conv*)

    apply text range table conversion to data

        Parameters data : numpy 1D array

            raw data to be converted to physical value

            conv : mdfinfo3.info3 conversion block ('CCBlock') dict

        Returns converted data to physical value

MDFINFO3 MODULE DOCUMENTATION

Measured Data Format blocks parser for version 3.x

Created on Thu Dec 9 12:57:28 2014

3.1 Platform and python version

With Unix and Windows for python 2.6+ and 3.2+

> **Author** Aymeric Rateau

3.2 Dependencies

- Python >2.6, >3.2 <http://www.python.org>
- Numpy >1.6 <http://numpy.scipy.org>

3.3 Attributes

PythonVersion [float] Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

3.4 mdfinfo3 module

class `mdfinfo3.`**`info3`**(*fileName=None*, *fid=None*, *filterChannelNames=False*)
> Bases: `dict`

> mdf file info class version 3.x MDFINFO is a class information about an MDF (Measure Data Format) file Based on following specification <http://powertrainnvh.com/nvh/MDFspecificationv03.pdf>

> **Notes**

> mdfinfo(FILENAME) contains a dict of structures, for each data group, containing key information about all channels in each group. FILENAME is a string that specifies the name of the MDF file. General dictionary structure is the following

> > •mdfinfo['HDBlock'] header block

- mdfinfo['DGBlock'][dataGroup] Data Group block

- mdfinfo['CGBlock'][dataGroup][channelGroup] Channel Group block

- mdfinfo['CNBlock'][dataGroup][channelGroup][channel] Channel block including text blocks for comment and identifier

- mdfinfo['CCBlock'][dataGroup][channelGroup][channel] Channel conversion information

Attributes

filterChannel-Names	(bool, optional) flag to filter long channel names including module names separated by a '.'
fileName	(str) name of file

Methods

static **blockformats3**(*block*, *version=0*)

This function returns all the predefined formats for the different blocks in the MDF file

> **Parameters** **block** : str
>
> > kind of block
>
> **version** : int
>
> > mdf version
>
> **Returns** nested list of str and int describing structure of block to be used by mdfblockread3 method

listChannels3(*fileName=None*)

reads data, channel group and channel blocks to list channel names

> **Returns** list of channel names

Attributes

fileName	(str) file name

static **mdfblockread3**(*blockFormat*, *fid*, *pointer*, *removeTrailing0=True*)

Extract block of data from MDF file in original data types. Returns a dictionary with keys specified in data structure blockFormat

> **Parameters** **blockFormat** : nested list
>
> > output of blockformats3 method
>
> **fid** : float
>
> > file identifier
>
> **pointer** : int
>
> > position of block in file
>
> **removeTrailing0** : bool, optional
>
> > removes or not the trailing 0 from strings
>
> **Returns** Block content in a dict

readinfo3 (*fid*)
read all file blocks except data

> **Parameters** **fid** : float
>
> > file identifier

MDF4READER MODULE DOCUMENTATION

Measured Data Format file reader module for version 4.x.

4.1 Platform and python version

With Unix and Windows for python 2.6+ and 3.2+

> **Author** Aymeric Rateau

Created on Thu Dec 10 12:57:28 2013

4.2 Dependencies

- Python >2.6, >3.2 <http://www.python.org>
- Numpy >1.6 <http://numpy.scipy.org>
- bitarray to parse bits in not aligned bytes
- Sympy to convert channels with formula if needed
- zlib to uncompress data block if needed

4.3 Attributes

PythonVersion [float] Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

4.4 mdf4reader module

class `mdf4reader.`**`DATA`**(*fid*, *pointer*)
> Bases: `dict`

> DATA class is organizing record classes itself made of recordchannel. This class inherits from dict. Keys are corresponding to channel group recordID A DATA class corresponds to a data block, a dict of record classes (one per channel group) Each record class contains a list of recordchannel class representing the structure of channel record.

Attributes

fid	(io.open) file identifier
pointerToData	(int) position of Data block in mdf file
type	(str) 'sorted' or 'unsorted' data block

Methods

addRecord(record)	Adds a new record in DATA class dict
read(channelList, zip=None)	Reads data block
load(record, zip=None, nameList=None)	Reads sorted data block from record definition
readRecord(recordID, buf, channelList=None):	read record from a buffer

addRecord(*record*)
> Adds a new record in DATA class dict.

> > **Parameters record class**

> > > channel group definition listing record channel classes

load(*record*, *zip=None*, *nameList=None*, *sortedFlag=True*)
> Reads data block from record definition

> > **Parameters record class**

> > > channel group definition listing record channel classes

> > **zip** : bool, optional

> > > flag to track if data block is compressed

> > **nameList** : list of str, optional

> > > list of channel names

> > **Returns** numpy recarray of data

read(*channelList*, *zip=None*)
> Reads data block

> > **Parameters channelList** : list of str

> > > list of channel names

> > **zip** : bool, optional

> > > flag to track if data block is compressed

readRecord(*recordID*, *buf*, *channelList=None*)
> read record from a buffer

> > **Parameters recordID** : int

> > > record identifier

> > **buf** : str

> > > buffer of data from file to be converted to channel raw data

> > **channelList** : list of str

> > > list of channel names to be read

mdf4reader.**DATABlock**(*record*, *parent_block*, *channelList=None*, *sortedFlag=True*)
    DATABlock converts raw data into arrays

> **Parameters record** : class
>
>> record class instance describing a channel group record
>
>> **parent_block** : class
>>
>>> MDFBlock class containing at least parent block header
>>
>> **channelList** : list of str, optional
>>
>>> defines list of channels to only read, can be slow but saves memory, for big files
>>
>> **sortedFlag** : bool, optional
>>
>>> flag to know if data block is sorted (only one Channel Group in block) or unsorted (several Channel Groups identified by a recordID). As unsorted block can contain CG records in random order, block is processed iteratively, not in raw like sorted -> much slower reading
>
> **Returns** a recarray containing the channels data

Notes

> This function will read DTBlock, RDBlock, DZBlock (compressed), RDBlock (VLSD), sorted or unsorted

mdf4reader.**append_field**(*rec*, *name*, *arr*, *numpy_dtype=None*)
    append new field in a recarray

> **Parameters rec** : numpy recarray
>
>> **name** : str
>>
>>> name of field to be appended
>>
>> **arr** : numpy array to be appended
>>
>> **numpy_dtype** : numpy dtype, optional
>>
>>> apply same dtype as arr by default but can be modified
>
> **Returns** numpy recarray appended

mdf4reader.**arrayformat4**(*signalDataType*, *numberOfBits*)
    function returning numpy style string from channel data type and number of bits

> **Parameters signalDataType** : int
>
>> channel data type according to specification
>
>> **numberOfBits** : int
>>
>>> number of bits taken by channel data in a record
>
> **Returns dataType** : str
>
>> numpy dtype format used by numpy.core.records to read channel raw data

mdf4reader.**bits_to_bytes**(*nBits*)
    Converts number of bits into number of aligned bytes

> **Parameters nBits** : int
>
>> number of bits

> > **Returns** number of equivalent bytes

mdf4reader.**change_field_name**(*arr*, *old_name*, *new_name*)
> modifies name of field in a recarray

> > **Parameters arr** : numpy recarray

> > > **old_name** : str

> > > > old field

> > > **new_name** : str

> > > > new field

> > **Returns** numpy recarray with modified field name

mdf4reader.**convertChannelData4**(*channel*, *channelName*, *convert_tables*, *multiProc=False*, *Q=None*)
> converts specific channel from raw to physical data according to CCBlock information

> > **Parameters channelName** : dict

> > > channel dict containing keys like 'data', 'unit', 'comment' and potentially 'conversion' dict

> > > **channelName** : str

> > > > name of channel

> > > **convert_tables** : bool

> > > > activates computation intensive loops for conversion with tables. Default is False

> > > **multiProc** : bool, default False

> > > > flag to put data in multiprocess queue

> > > **Q** : Queue class, default None

> > > > Queue used for multiprocessing

> > **Returns** dict

> > > returns dict with channelName key containing numpy array converted to physical values according to conversion type

mdf4reader.**convertName**(*channelName*)
> Adds '_title' to channel name for numpy.core.records methods.

mdf4reader.**datatypeformat4**(*signalDataType*, *numberOfBits*)
> function returning C format string from channel data type and number of bits

> > **Parameters signalDataType** : int

> > > channel data type according to specification

> > > **numberOfBits** : int

> > > > number of bits taken by channel data in a record

> > **Returns dataType** : str

> > > C format used by fread to read channel raw data

mdf4reader.**equalizeStringLength**(*buf*)
> Makes all strings in a list having same length by appending spaces strings.

> > **Parameters buf** : list of str

Returns list of str elements all having same length

mdf4reader.**formulaConv**(*vect*, *formula*)

apply formula conversion to data

Parameters **vect** : numpy 1D array

raw data to be converted to physical value

cc_val : mdfinfo4.info4 conversion block ('CCBlock') dict

Returns converted data to physical value

class mdf4reader.**invalid_bytes**(*info*, *dataGroup*, *channelGroup*, *recordIDsize*, *byte_aligned=True*)

invalid_bytes class to handle invalid bytes in record if existing

Attributes

name	(str) Name of channel
signalDataType	(int) signal type according to specification
bitCount	(int) number of bits used to store channel record
nBytes	(int) number of bytes (1 byte = 8 bits) taken by channel record
dataFormat	(str) numpy dtype as string
Format :	C format understood by fread
CFormat	(struct class instance) struct instance to convert from C Format
byteOffset	(int) position of channel record in complete record in bytes
bitOffset	(int) bit position of channel value inside byte in case of channel having bit count below 8
RecordFormat	(list of str) dtype format used for numpy.core.records functions ((name,name_title),str_stype)
channelType	(int) channel type
posByteBeg	(int) start position in number of bit of channel record in complete record
posByteEnd	(int) end position in number of bit of channel record in complete record
posBitBeg	(int) start position in number of bit of channel record in complete record
posBitEnd	(int) end position in number of bit of channel record in complete record
VLSD_CG_Flag	(bool) flag when Channel Group VLSD is used
data	(int) pointer to data block linked to a channel (VLSD, MLSD)

Methods

__init__(info, dataGroup, channelGroup, recordIDsize)	constructor
channel_validity(channelName)	returns channel validity bit array

validity_channel(*channelName*)

extract channel validity bits

Parameters **channelName** : str

channel name

mdf4reader.**linearConv**(*vect*, *cc_val*)

apply linear conversion to data

Parameters **vect** : numpy 1D array

raw data to be converted to physical value

cc_val : mdfinfo4.info4 conversion block ('CCBlock') dict

> **Returns** converted data to physical value

class `mdf4reader.`**`mdf4`**(*fileName=None*, *info=None*, *multiProc=False*, *channelList=None*, *convertAfter-Read=True*)

Bases: `dict`

mdf file reader class from version 4.0 to 4.1

Attributes

fileName	(str) file name
Version-Number	(int) mdf file version number
master-Channel-List	(dict) Represents data structure: a key per master channel with corresponding value containing a list of channels One key or master channel represents then a data group having same sampling interval.
multiProc	(bool) Flag to request channel conversion multi processed for performance improvement. One thread per data group.
con-vertAfter-Read	(bool) flag to convert raw data to physical just after read
filterChan-nelNames	(bool) flag to filter long channel names from its module names separated by '.'
author	(str)
organisa-tion	(str)
project	(str)
subject	(str)
comment	(str)
time	(str)
date	(str)

Methods

read4(fileName=None, info=None, multiProc=False, channelList=None, convertAfterRead=True)	Reads mdf 4.x file data and stores it in dict
getChannelData4(channelName)	Returns channel numpy array
convertChannel4(channelName)	converts specific channel from raw to physical data according to CCBlock information
convertAllChannel4()	Converts all channels from raw data to converted data according to CCBlock information

`convertAllChannel4`()

Converts all channels from raw data to converted data according to CCBlock information Converted data will take more memory.

`convertChannel4`(*channelName*)

converts specific channel from raw to physical data according to CCBlock information

> **Parameters channelName** : str
>
> > Name of channel

`getChannelData4`(*channelName*)

Returns channel numpy array

Parameters **channelName** : str

> channel name

Notes

This method is the safest to get channel data as numpy array from 'data' dict key might contain raw data

read4 (*fileName=None*, *info=None*, *multiProc=False*, *channelList=None*, *convertAfterRead=True*)
Reads mdf 4.x file data and stores it in dict

> **Parameters** **fileName** : str, optional
>
> > file name
>
> **info** : mdfinfo4.info4 class
>
> > info3 class containing all MDF Blocks
>
> **multiProc** : bool
>
> > flag to activate multiprocessing of channel data conversion
>
> **channelList** : list of str, optional
>
> > list of channel names to be read If you use channelList, reading might be much slower but it will save you memory. Can be used to read big files
>
> **convertAfterRead** : bool, optional
>
> > flag to convert channel after read, True by default If you use convertAfterRead by setting it to false, all data from channels will be kept raw, no conversion applied. If many float are stored in file, you can gain from 3 to 4 times memory footprint To calculate value from channel, you can then use method .getChannelData()

mdf4reader.**processDataBlocks4** (*Q*, *buf*, *info*, *dataGroup*, *channelList*, *multiProc*)
Put raw data from buf to a dict L and processes nested nBit channels

> **Parameters** **Q** : multiprocessing.Queue, optional
>
> > Queue for multiprocessing
>
> **buf** : DATA class
>
> > contains raw data
>
> **info** : info class
>
> > contains infomation from MDF Blocks
>
> **dataGroup** : int
>
> > data group number according to info class
>
> **channelList** : list of str, optional
>
> > list of channel names to be processed
>
> **multiProc** : bool
>
> > flag to return Queue or dict
>
> **Returns** **Q** : multiprocessing.Queue
>
> > updates Queue containing L dict
>
> **L** : dict

dict of channels

mdf4reader.**rationalConv**(*vect*, *cc_val*)

apply rational conversion to data

> **Parameters vect** : numpy 1D array
>
> > raw data to be converted to physical value
>
> **cc_val** : mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **Returns** converted data to physical value

class mdf4reader.**record**(*dataGroup*, *channelGroup*)

Bases: list

record class lists recordchannel classes, it is representing a channel group

Attributes

CGrecordLength	(int) length of record corresponding of channel group in Byte CG Block information
recordLength	(int) length of record as understood by program based on C datatypes
numberOfRecords	(int) number of records in data block
recordID	(int) recordID corresponding to channel group
recordIDsize	(int) size of recordID
recordIDCFormat	(str) record identifier C format string as understood by fread
dataGroup	(int:) data group number
channelGroup	(int) channel group number
numpyDataRecordFor-mat	(list) list of numpy (dtype) for each channel
dataRecordName	(list) list of channel names used for recarray attribute definition
master	(dict) define name and number of master channel
recordToChannelMatch-ing	(dict) helps to identify nested bits in byte
channelNames	(list) channel names to be stored, useful for low memory consumption but slow
Flags	(bool) channel flags as from specification
VLSD_CG	(dict) dict of Channel Group VLSD, key being recordID
VLSD	(list of recordChannel) list of recordChannel being VLSD
MLSD	(dict) copy from info['MLSD'] if existing
byte_aligned	(Bool) flag for byte aligned record
invalid_channel	(Default None) invalid_byte class if existing in record otherwise None

Methods

addChannel(info, channelNumber)	
loadInfo(info)	
readSortedRecord(fid, pointer, channelList=None)	
readRecordBuf(buf, channelList=None)	
readBitarray(bita, channelList=None)	

addChannel(*info*, *channelNumber*)

add a channel in class

> **Parameters info** : mdfinfo4.info4 class

> > **channelNumber** : int
> >
> > > channel number in mdfinfo4.info4 class

> **loadInfo**(*info*)
> > gathers records related from info class
> >
> > > **Parameters info** : mdfinfo4.info4 class

> **readBitarray**(*bita*, *channelList=None*)
> > reads stream of record bytes using bitarray module needed for not byte aligned data
> >
> > > **Parameters bitarray** : stream
> > >
> > > > stream of bytes
> > >
> > > **channelList** : List of str, optional
> > >
> > > > list of channel to read
> > >
> > > **Returns rec** : numpy recarray
> > >
> > > > contains a matrix of raw data in a recarray (attributes corresponding to channel name)

> **readRecordBuf**(*buf*, *channelList=None*)
> > read stream of record bytes
> >
> > > **Parameters buf** : stream
> > >
> > > > stream of bytes read in file
> > >
> > > **channelList** : list of str, optional
> > >
> > > > list of channel to read
> > >
> > > **Returns rec** : dict
> > >
> > > > # returns dictionary of channel with its corresponding values

> **readSortedRecord**(*fid*, *pointer*, *channelList=None*)
> > reads record, only one channel group per datagroup Parameters —————- fid : float
> >
> > > file identifier
> >
> > **pointer** position in file of data block beginning
> >
> > **channelList** [list of str, optional] list of channel to read
> >
> > > **Returns rec** : numpy recarray
> > >
> > > > contains a matrix of raw data in a recarray (attributes corresponding to channel name)

> > **Notes**

> > If channelList is None, read data using numpy.core.records.fromfile that is rather quick. However, in case of large file, you can use channelList to load only interesting channels or only one channel on demand, but be aware it might be much slower.

class mdf4reader.**recordChannel**(*info*, *dataGroup*, *channelGroup*, *channelNumber*, *recordIDsize*)
> recordChannel class gathers all about channel structure in a record

Attributes

name	(str) Name of channel
channelNumber	(int) channel number corresponding to mdfinfo3.info3 class
signalDataType	(int) signal type according to specification
bitCount	(int) number of bits used to store channel record
nBytes	(int) number of bytes (1 byte = 8 bits) taken by channel record
dataFormat	(str) numpy dtype as string
Format :	C format understood by fread
CFormat	(struct class instance) struct instance to convert from C Format
byteOffset	(int) position of channel record in complete record in bytes
bitOffset	(int) bit position of channel value inside byte in case of channel having bit count below 8
RecordFormat	(list of str) dtype format used for numpy.core.records functions ((name,name_title),str_stype)
channelType	(int) channel type
posByteBeg	(int) start position in number of byte of channel record in complete record
posByteEnd	(int) end position in number of byte of channel record in complete record
posBitBeg	(int) start position in number of bit of channel record in complete record
posBitEnd	(int) end position in number of bit of channel record in complete record
VLSD_CG_Flag	(bool) flag when Channel Group VLSD is used
data	(int) pointer to data block linked to a channel (VLSD, MLSD)

Methods

__init__(info, dataGroup, channelGroup, channelNumber, recordIDsize)	constructor
__str__()	to print class attributes

mdf4reader.**textToTextConv**(*vect*, *cc_ref*)
> apply text to text conversion to data

> > **Parameters** **vect** : numpy 1D array

> > > raw data to be converted to physical value

> > **cc_ref** : cc_ref from mdfinfo4.info4 conversion block ('CCBlock') dict

> > **Returns** converted data to physical value

mdf4reader.**textToValueConv**(*vect*, *cc_val*, *cc_ref*)
> apply text to value conversion to data

> > **Parameters** **vect** : numpy 1D array

> > > raw data to be converted to physical value

> > **cc_val** : cc_val from mdfinfo4.info4 conversion block ('CCBlock') dict

> > **cc_ref** : cc_ref from mdfinfo4.info4 conversion block ('CCBlock') dict

> > **Returns** converted data to physical value

mdf4reader.**valueRangeToTextConv**(*vect*, *cc_val*, *cc_ref*)
> apply value range to text conversion to data

> > **Parameters** **vect** : numpy 1D array

> > > raw data to be converted to physical value

> **cc_val** : cc_val from mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **cc_ref** : cc_ref from mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **Returns** converted data to physical value

mdf4reader.**valueRangeToValueTableConv**(*vect*, *cc_val*)
    apply value range to value table conversion to data

> **Parameters** **vect** : numpy 1D array
>
> > raw data to be converted to physical value
>
> **cc_val** : mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **Returns** converted data to physical value

mdf4reader.**valueToTextConv**(*vect*, *cc_val*, *cc_ref*)
    apply value to text conversion to data

> **Parameters** **vect** : numpy 1D array
>
> > raw data to be converted to physical value
>
> **cc_val** : cc_val from mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **cc_ref** : cc_ref from mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **Returns** converted data to physical value

mdf4reader.**valueToValueTableWInterpConv**(*vect*, *cc_val*)
    apply value to value table with interpolation conversion to data

> **Parameters** **vect** : numpy 1D array
>
> > raw data to be converted to physical value
>
> **cc_val** : mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **Returns** converted data to physical value

mdf4reader.**valueToValueTableWOInterpConv**(*vect*, *cc_val*)
    apply value to value table without interpolation conversion to data

> **Parameters** **vect** : numpy 1D array
>
> > raw data to be converted to physical value
>
> **cc_val** : mdfinfo4.info4 conversion block ('CCBlock') dict
>
> **Returns** converted data to physical value

MDFINFO4 MODULE DOCUMENTATION

Measured Data Format blocks paser for version 4.x

5.1 Platform and python version

With Unix and Windows for python 2.6+ and 3.2+

Created on Sun Dec 15 12:57:28 2013

> **Author** Aymeric Rateau

5.2 Dependencies

- Python >2.6, >3.2 <http://www.python.org>
- Numpy >1.6 <http://numpy.scipy.org>

5.3 Attributes

PythonVersion [float] Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

5.4 mdfinfo4 module

class mdfinfo4.**ATBlock** (*fid*, *pointer*)
    Bases: mdfinfo4.MDFBlock

    reads Attachment block and saves in class dict

    Methods

class mdfinfo4.**CABlock** (*fid*, *pointer*)
    Bases: mdfinfo4.MDFBlock

    reads Channel Array block and saves in class dict

Methods

class mdfinfo4.**CCBlock** (*fid*, *pointer*)
    Bases: `mdfinfo4.MDFBlock`

    reads Channel Conversion block and saves in class dict

Methods

class mdfinfo4.**CGBlock** (*fid*, *pointer*)
    Bases: `mdfinfo4.MDFBlock`

    reads Channel Group block and saves in class dict

Methods

class mdfinfo4.**CHBlock** (*fid*, *pointer*)
    Bases: `mdfinfo4.MDFBlock`

    reads Channel Hierarchy block and saves in class dict

Methods

class mdfinfo4.**CNBlock** (*fid*, *pointer*)
    Bases: `mdfinfo4.MDFBlock`

    reads Channel block and saves in class dict

Methods

class mdfinfo4.**CommentBlock** (*fid*, *pointer*, *MDType=None*)
    Bases: `mdfinfo4.MDFBlock`

    reads Comment block and saves in class dict

Notes

Can read xml (MD metadata) or text (TX) comments from several kind of blocks

Methods

extractXmlField (*xml_tree*, *field*)
    Extract Xml field from a xml tree

        Parameters **xml_tree** : xml tree from xml.etree.ElementTree

            field : str

        Returns field value in xml tree

class mdfinfo4.**DGBlock** (*fid*, *pointer*)
    Bases: `mdfinfo4.MDFBlock`

    reads Data Group block and saves in class dict

    ### Methods

class mdfinfo4.**EVBlock** (*fid*, *pointer*)
    Bases: `mdfinfo4.MDFBlock`

    reads Event block and saves in class dict

    ### Methods

class mdfinfo4.**FHBlock** (*fid*, *pointer*)
    Bases: `mdfinfo4.MDFBlock`

    reads File History block and save in class dict

    ### Methods

class mdfinfo4.**HDBlock** (*fid*, *pointer=64*)
    Bases: `mdfinfo4.MDFBlock`

    reads Header block and save in class dict

    ### Methods

class mdfinfo4.**IDBlock** (*fid*)
    Bases: `mdfinfo4.MDFBlock`

    reads ID Block and save in class dict

    ### Methods

class mdfinfo4.**MDFBlock**
    Bases: `dict`

    MDFBlock base class for the MDF related block classes

    ### Methods

loadHeader(fid, pointer)	reads block's header and put in class dict
mdfblockread(fid, type, count)	converts a byte array of length count to a given data type
mdfblockreadCHAR(fid, count)	reads a character chain of length count encoded in latin.
mdfblockreadBYTE(fid, count)	reads an array of UTF-8 encoded bytes

    loadHeader (*fid*, *pointer*)
        reads block's header and put in class dict

            Parameters fid : float

                file identifier

5.4. mdfinfo4 module

> > > **pointer** : int
> > >
> > > > position of block in file
> >
> > static **mdfblockread**(*fid*, *type*, *count*)
> > > converts a byte array of length count to a given data type
> > >
> > > > **Parameters type** : str
> > > >
> > > > > C format data type
> > > >
> > > > **count** : int
> > > >
> > > > > number of elements to sequentially read
> > > >
> > > > **Returns** array of values of 'type' parameter
> >
> > static **mdfblockreadBYTE**(*fid*, *count*)
> > > reads an array of UTF-8 encoded bytes. Removes trailing 0
> > >
> > > > **Parameters count** : int
> > > >
> > > > > number of bytes to read
> > > >
> > > > **Returns** bytes array of length count
> >
> > static **mdfblockreadCHAR**(*fid*, *count*)
> > > reads a character chain of length count encoded in latin. Removes trailing 0
> > >
> > > > **Parameters count** : int
> > > >
> > > > > number of characters to read
> > > >
> > > > **Returns** a string of length count

class mdfinfo4.**SIBlock**(*fid*, *pointer*)
> Bases: mdfinfo4.MDFBlock
>
> reads Source Information block and saves in class dict

Methods

class mdfinfo4.**SRBlock**(*fid*, *pointer*)
> Bases: mdfinfo4.MDFBlock
>
> reads Sample Reduction block and saves in class dict

Methods

mdfinfo4.**elementTreeToDict**(*element*)
> converts xml tree into dictionnary
>
> > **Parameters element** : xml tree from xml.etree.ElementTree
> >
> > **Returns** dict of xml tree flattened

class mdfinfo4.**info4**(*fileName=None*, *fid=None*)
> Bases: dict
>
> information block parser fo MDF file version 4.x

Notes

mdfinfo(FILENAME) contains a dict of structures, for each data group, containing key information about all channels in each group. FILENAME is a string that specifies the name of the MDF file. Either file name or fid should be given. General dictionary structure is the following

- mdfinfo['HDBlock'] header block

- mdfinfo['DGBlock'][dataGroup] Data Group block

- mdfinfo['CGBlock'][dataGroup][channelGroup] Channel Group block

- mdfinfo['CNBlock'][dataGroup][channelGroup][channel] Channel block including text blocks for comment and identifier

- mdfinfo['CCBlock'][dataGroup][channelGroup][channel] Channel conversion information

Attributes

fileName	(str) name of file

Methods

listChannels4 (*fileName=None*)
 Read MDF file and extract its complete structure

> **Parameters fileName** : str
>
>> file name
>
> **Returns** list of channel names contained in file

readATBlock (*selfself*, *fid*, *pointer*)
 reads Attachment blocks

> **Parameters fid** : float
>
>> file identifier
>
>> **pointer** : int
>
>> position of ATBlock in file
>
> **Returns** Attachments Blocks in a dict

readCGBlock (*fid*, *dg*, *channelNameList=False*)
 reads Channel Group blocks

> **Parameters fid** : float
>
>> file identifier
>
>> **dg** : int
>
>> data group number
>
>> **channelNameList** : bool
>
>> Flag to reads only channel blocks for listChannels4 method

readCNBlock (*fid*, *dg*, *cg*, *channelNameList=False*)
 reads Channel blocks

> > > **Parameters fid** : float
> > >
> > > > file identifier
> > >
> > > **dg** : int
> > >
> > > > data group number
> > >
> > > **cg** : int
> > >
> > > > channel group number in data group
> > >
> > > **channelNameList** : bool
> > >
> > > > Flag to reads only channel blocks for listChannels4 method

readComposition(*fid*, *dg*, *cg*, *MLSDChannels*, *channelNameList=False*)
> check for composition of channels, arrays or structures

> > **Parameters fid** : float
> >
> > > file identifier
> >
> > **dg** : int
> >
> > > data group number
> >
> > **cg** : int
> >
> > > channel group number in data group
> >
> > **MLSDChannels** : list of int
> >
> > > channel numbers
> >
> > **channelNameList** : bool
> >
> > > Flag to reads only channel blocks for listChannels4 method
> >
> > **Returns** MLSDChannels list of appended Maximum Length Sampling Data channels

readDGBlock(*fid*, *channelNameList=False*)
> reads Data Group Blocks

> > **Parameters fid** : float
> >
> > > file identifier
> >
> > **channelNameList** : bool
> >
> > > Flag to reads only channel blocks for listChannels4 method

readSRBlock(*fid*, *pointer*)
> reads Sample Reduction Blocks

> > **Parameters fid** : float
> >
> > > file identifier
> >
> > **pointer** : int
> >
> > > position of SRBlock in file
> >
> > **Returns** Sample Reduction Blocks in a dict

readinfo(*fid*)
> read all file blocks except data

> > **Parameters fid** : float

file identifier

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

m

A

B

C

D

E

F

G

H

I

K

L