mdfreader Documentation

Release 2.8

Aymeric Rateau

Nov 26, 2018

CONTENTS

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

mdfreader.mdfreader.**PythonVersion**
> *float* – Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

1.3 mdfreader

class mdfreader.mdfreader.**Mdf** (*file_name=None*, *channel_list=None*, *convert_after_read=True*, *filter_channel_names=False*, *no_data_loading=False*, *compression=False*, *convert_tables=False*, *metadata=2*)
> Bases: *mdfreader.mdf3reader.Mdf3*, *mdfreader.mdf4reader.Mdf4*

> Mdf class

fileName
> *str* – file name

MDFVersionNumber
> *int* – mdf file version number

masterChannelList
> *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.

fileMetadata
> *dict* – file metadata with minimum keys : author, organisation, project, subject, comment, time, date

read(file_name = None, multi_processed = False, channel_list=None, convert_after_read

filter_channel_names=False, no_data_loading=False, compression=False)
> reads mdf file version 3.x and 4.x

write (*file_name=None*)
> writes simple mdf file

get_channel_data (*channel_name*)
> returns channel numpy array

convert_all_channel ()
> converts all channel data according to CCBlock information

get_channel_unit (*channel_name*)
> returns channel unit

plot (*channels*)
> Plot channels with Matplotlib

resample (*sampling_time = 0.1*, *master_channel=None*)
> Resamples all data groups

export_to_csv (*file_name = None*, *sampling = 0.1*)
> Exports mdf data into CSV file

export_to_NetCDF (*file_name = None*, *sampling = None*)
> Exports mdf data into netcdf file

export_to_hdf5 (*file_name = None*, *sampling = None*)
> Exports mdf class data structure into hdf5 file

export_to_matlab (*file_name = None*)
> Exports mdf class data structure into Matlab file

export_to_excel (*file_name = None*)
> Exports mdf data into excel 95 to 2003 file

export_to_xlsx (*file_name=None*)
> Exports mdf data into excel 2007 and 2010 file

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

keep_channels (*channel_list*)
> keeps only list of channels and removes the other channels

---

merge_mdf(mdf_class):
    Merges data of 2 mdf classes

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
# list channels grouped by raster or master channel
>>> yop.masterChannelList
>>> yop.plot('channelName') or yop.plot({'channel1','channel2'})
>>> yop.resample(0.1) or yop.resample()
>>> yop.export_to_csv(sampling=0.01)
>>> yop.export_to_NetCDF()
>>> yop.export_to_hdf5()
>>> yop.export_to_matlab()
>>> yop.export_to_excel()
>>> yop.export_to_xlsx()
>>> yop.convert_to_pandas() # converts data groups into pandas dataframes
>>> yop.write() # writes mdf file
# drops all the channels except the one in argument
>>> yop.keep_channels({'channel1','channel2','channel3'})
>>> yop.get_channel_data('channelName') # returns channel numpy array
```

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

convert_to_pandas(*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

cut(*begin=None*, *end=None*)
    Cut data

        Parameters

- **begin** (*float*) – beginning value in master channel from which to start cutting in all channels

- **end** (*float*) – ending value in master channel from which to start cutting in all channels

Notes

Use this method if whole data in mdf are using same physical or type of master channel (for instance time).

export_to_NetCDF (*file_name=None*, *sampling=None*)
   Exports mdf data into netcdf file

   Parameters

- **file_name** (*str, optional*) – file name. If no name defined, it will use original mdf name and path

- **sampling** (*float, optional*) – sampling interval.

Notes

Dependency: scipy

export_to_csv (*file_name=None*, *sampling=None*)
   Exports mdf data into CSV file

   Parameters

- **file_name** (*str, optional*) – file name. If no name defined, it will use original mdf name and path

- **sampling** (*float, optional*) – sampling interval. None by default

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

export_to_excel (*file_name=None*)
   Exports mdf data into excel 95 to 2003 file

   Parameters file_name (*str, optional*) – file name. If no name defined, it will use original mdf name and path

Notes

xlwt is not fast 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 2003 is becoming rare these days, prefer using exportToXlsx Dependencies: xlwt for python 2.6+, xlwt3 for python 3.2+

export_to_hdf5 (*file_name=None*, *sampling=None*, *compression=None*, *compression_opts=None*)
   Exports mdf class data structure into hdf5 file

   Parameters

- **file_name** (*str, optional*) – file name. If no name defined, it will use original mdf name and path

- **sampling** (*float, optional*) – sampling interval.

- **compression** (*str, optional*) – HDF5 compression algorithm. Valid options are 'gzip', 'lzf'. gzip compression recommended for portability. szip compression not supported due to legal reasons.

- **compression_opts** (*int, optional*) – HDF5 gzip compression level, 0-9. Only valid if gzip compression is used. Level 4 (default) recommended for best balance between compression and time.

Notes

The maximum attributes will be stored Data structure will be similar has it is in masterChannelList attribute Dependency: h5py

export_to_matlab (*file_name=None*)

Export mdf data into Matlab file format 5, tentatively compressed

> **Parameters file_name** (*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 Dependency: scipy

export_to_xlsx (*file_name=None*)

Exports mdf data into excel 2007 and 2010 file

> **Parameters file_name** (*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 Dependency: openpyxl

get_channel_data (*channel_name*, *raw_data=False*)

Return channel numpy array

> **Parameters**
>
> - **channel_name** (*str*) – channel name
>
> - **raw_data** (*bool*) – flag to return non converted data
>
> **Returns** converted, if not already done, data corresponding to channel name
>
> **Return type** numpy array

Notes

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

keep_channels (*channel_list*)

keeps only list of channels and removes the other channels

> **Parameters channel_list** (*list of str*) – list of channel names

merge_mdf (*mdf_class*)

Merges data of 2 mdf classes

> **Parameters mdf_class** (*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(*channel_name_list_of_list*)

Plot channels with Matplotlib

> **Parameters channel_name_list_of_list**(*str or list of str or list of list of str*) – channel name or list of channel names or list of list of channel names list of list will create multiplots

Notes

Channel description and unit will be tentatively displayed with axis labels

plot_all()

read(*file_name=None*, *multi_processed=False*, *channel_list=None*, *convert_after_read=True*, *filter_channel_names=False*, *no_data_loading=False*, *compression=False*, *metadata=2*)

reads mdf file version 3.x and 4.x

> **Parameters**
>
> - **file_name** (*str, optional*) – file name
>
> - **multi_processed** (*bool*) – flag to activate multiprocessing of channel data conversion
>
> - **channel_list** (*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
>
> - **convert_after_read** (*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()
>
> - **filter_channel_names** (*bool, optional*) – flag to filter long channel names from its module names separated by '.'
>
> - **no_data_loading** (*bool, optional*) – Flag to read only file info but no data to have minimum memory use
>
> - **compression** (*bool or str, optional*) – To compress data in memory using blosc or bcolz, takes cpu time if compression = int(1 to 9), uses bcolz for compression if compression = 'blosc', uses blosc for compression Choice given, efficiency depends of data
>
> - **metadata** (*int, optional, default = 2*) – Reading metadata has impact on performance, especially for mdf 4.x using xml. 2: minimal metadata reading (mostly channel blocks) 1: used for noDataLoading 0: all metadata reading, including Source Information, Attachment, etc..

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

> **Warning:** MultiProc use should be avoided when reading several files in a batch, it is not thread safe. You should better multi process instances of mdf rather than using multiproc in mdf class (see implementation of mdfconverter)

resample(*sampling_time=None*, *master_channel=None*)
: Resamples all data groups into one data group having defined sampling interval or sharing same master channel

    Parameters

    - **sampling_time** (`float, optional`) – resampling interval, None by default. If None, will merge all datagroups into a unique datagroup having the highest sampling rate from all datagroups

    - ****or**** –

    - **master_channel** (`str, optional`) – master channel name to be 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(*file_name=None*, *compression=False*)
: Writes simple mdf file, same format as originally read, default is 4.x

    Parameters

    - **file_name** (`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

    - **compression** (`bool`) – Flag to store data compressed (from mdf version 4.1) If activated, will write in version 4.1 even if original file is in version 3.x

    ### Notes

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

class mdfreader.mdfreader.**MdfInfo**() -> *new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)*

: Bases: dict

    fid

    fileName

    filterChannelNames

    list_channels(*file_name=None*)
    : Read MDF file blocks and returns a list of contained channels

---

> > **Parameters file_name** (*string*) – file name
>
> > **Returns nameList** – list of channel names
>
> > **Return type** list of string

mdfversion

read_info (*file_name=None*, *fid=None*, *minimal=0*)
> Reads MDF file and extracts its complete structure

> > **Parameters**
> >
> > - **file_name** (*str, optional*) – file name. If not input, uses fileName attribute
> >
> > - **fid** (*file identifier, optional*) –
> >
> > - **minimal** (*int*) – 0 will load every metadata 1 will load DG, CG, CN and CC 2 will load only DG

zipfile

MDF MODULE DOCUMENTATION

mdf_skeleton module describing basic mdf structure and methods

Created on Thu Sept 24 2015

> **Author** Aymeric Rateau

2.1 Dependencies

- Python >2.6, >3.2 <http://www.python.org>

- Numpy >1.6 <http://numpy.scipy.org>

2.2 mdf

class `mdfreader.mdf.`**`CompressedData`**
Bases: `object`

`compression`(*a*)
data compression method

> **Parameters a** (`numpy array`) – data to be compresses

`data`

`decompression`()
data decompression

> **Returns**

> **Return type** uncompressed numpy array

`dtype`

class `mdfreader.mdf.`**`MdfSkeleton`**() -> *new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)*

Bases: `dict`

`MDFVersionNumber`

`add_channel`(*channel_name*, *data*, *master_channel*, *master_type=1*, *unit="*, *description="*, *conversion=None*, *info=None*, *compression=False*, *identifier=None*)
adds channel to mdf dict.

Parameters

- **channel_name** (`str`) – channel name

- **data** (`numpy array`) – numpy array of channel's data

- **master_channel** (`str`) – master channel name

- **master_type** (`int, optional`) – master channel type : 0=None, 1=Time, 2=Angle, 3=Distance, 4=index

- **unit** (`str, optional`) – unit description

- **description** (`str, optional`) – channel description

- **conversion** (`info class, optional`) – conversion description from info class

- **info** (`info class for CNBlock, optional`) – used for CABlock axis creation and channel conversion

- **compression** (`bool`) – flag to ask for channel data compression

- **identifier** (`tuple`) – tuple of int and str following below structure: (data group number, channel group number, channel number), (channel name, channel source, channel path), (group name, group source, group path)

add_metadata (*author=", organisation=", project=", subject=", comment=", date=", time="*)
adds basic metadata to mdf class

Parameters

- **author** (`str`) – author of file

- **organisation** (`str`) – organisation of author

- **project** (`str`) –

- **subject** (`str`) –

- **comment** (`str`) –

- **date** (`str`) –

- **time** (`str`) –

Notes

All fields are optional, default being empty string

convertAfterRead

convertTables

copy ()
copy a mdf class

> **Returns** **mdf_skeleton** – copy of a mdf_skeleton class
>
> **Return type** class instance

fid

fileMetadata

fileName

filterChannelNames

get_channel(*channel_name*)
Extract channel dict from mdf structure

> **Parameters channel_name** (*str*) – channel name
>
> **Returns**
>
> **Return type** channel dictionnary containing data, description, unit, etc.

get_channel_conversion(*channel_name*)
Extract channel conversion dict from mdf structure

> **Parameters channel_name** (*str*) – channel name
>
> **Returns**
>
> **Return type** channel conversion dict

get_channel_desc(*channel_name*)
Extract channel description information from mdf structure

> **Parameters channel_name** (*str*) – channel name
>
> **Returns**
>
> **Return type** channel description string

get_channel_master(*channel_name*)
Extract channel master name from mdf structure

> **Parameters channel_name** (*str*) – channel name
>
> **Returns**
>
> **Return type** channel master name string

get_channel_master_type(*channel_name*)
Extract channel master type information from mdf structure

> **Parameters channel_name** (*str*) – channel name
>
> **Returns channel mater type integer**
>
> **Return type** 0=None, 1=Time, 2=Angle, 3=Distance, 4=index

get_channel_unit(*channel_name*)
Returns channel unit string Implemented for a future integration of pint

> **Parameters channel_name** (*str*) – channel name
>
> **Returns** unit string description
>
> **Return type** str

get_invalid_bit(*channel_name*)

get_invalid_channel(*channel_name*)

info

masterChannelList

multiProc

remove_channel(*channel_name*)
removes channel from mdf dict.

> **Parameters channel_name** (*str*) – channel name

> **Returns**
>
> **Return type** value of mdf dict key=channel_name

remove_channel_conversion(*channel_name*)
  removes conversion key from mdf channel dict.

> **Parameters channel_name** (*str*) – channel name
>
> **Returns**
>
> **Return type** removed value from dict

rename_channel(*channel_name*, *new_name*)
  Modifies name of channel

> **Parameters**
>
> - **channel_name** (*str*) – channel name
> - **new_name** (*str*) – new channel name

set_channel_attachment(*channel_name*, *attachment*)
  Modifies channel attachment

> **Parameters**
>
> - **channel_name** (*str*) – channel name
> - **attachment** – channel attachment

set_channel_conversion(*channel_name*, *conversion*)
  Modifies conversion dict of channel

> **Parameters**
>
> - **channel_name** (*str*) – channel name
> - **conversion** (*dict*) – conversion dictionary

set_channel_data(*channel_name*, *data*, *compression=False*)
  Modifies data of channel

> **Parameters**
>
> - **channel_name** (*str*) – channel name
> - **data** (*numpy array*) – channel data
> - **compression** (*bool or str*) – trigger for data compression

set_channel_desc(*channel_name*, *desc*)
  Modifies description of channel

> **Parameters**
>
> - **channel_name** (*str*) – channel name
> - **desc** (*str*) – channel description

set_channel_master(*channel_name*, *master*)
  Modifies channel master name

> **Parameters**
>
> - **channel_name** (*str*) – channel name
> - **master** (*str*) – master channel name

set_channel_master_type(*channel_name*, *master_type*)
Modifies master channel type

> **Parameters**
>
> > • **channel_name** (`str`) – channel name
> >
> > • **master_type** (`int`) – master channel type

set_channel_unit(*channel_name*, *unit*)
Modifies unit of channel

> **Parameters**
>
> > • **channel_name** (`str`) – channel name
> >
> > • **unit** (`str`) – channel unit

set_invalid_bit(*channel_name*, *bit_position*)
returns the invalid bit position of channel

> **Parameters**
>
> > • **channel_name** (`str`) – channel name
> >
> > • **bit_position** – invalid bit position of channel within invalid channel bytes
>
> **Returns**
>
> **Return type** bit position

set_invalid_channel(*channel_name*, *invalid_channel*)

zipfile

MDF3READER MODULE DOCUMENTATION

Measured Data Format file reader module for version 3.x

Author Aymeric Rateau

Created on Sun Oct 10 12:57:28 2010

mdfreader.mdf3reader.**PythonVersion**
> *float* – Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

3.1 mdf3reader

class mdfreader.mdf3reader.**DATA**(*fid*, *pointer*)
> Bases: dict
>
> DATA class is organizing record classes itself made of channel. 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 channel class representing the structure of channel record.
>
> **fid**
> > *io.open* – file identifier
>
> **pointerToData**
> > *int* – position of Data block in mdf file
>
> **BlockLength**
> > *int* – total size of data block
>
> **add_record**(*record*)
> > Adds a new record in DATA class dict
>
> **read**(*channelSet*)
> > Reads data block
>
> **load_sorted**(*record*, *nameList=None*)
> > Reads sorted data block from record definition
>
> **load_unsorted**(*nameList=None*)
> > Reads unsorted data block, not yet implemented
>
> **add_record**(*record*)
> > Adds a new record in DATA class dict
> >
> > > **Parameters** **record** (*class*) – channel group definition listing record channel classes
>
> **load_sorted**(*record*, *name_list=None*)
> > Reads sorted data block from record definition
> >
> > > **Parameters**

- **record** (`class`) – channel group definition listing record channel classes

- **name_list** (`set of str, optional`) – list of channel names

> **Returns**

> **Return type** numpy recarray of data

load_unsorted (*name_list=None*)
> Reads unsorted data block from record definition

> **Parameters name_list** (`set of str, optional`) – list of channel names

> **Returns**

> **Return type** numpy recarray of data

read (*channel_set*, *file_name*)
> Reads data block

> **Parameters**

- **channel_set** (`set of str, optional`) – list of channel names

- **file_name** (`str`) – name of file

class mdfreader.mdf3reader.**Mdf3** (*file_name=None*, *channel_list=None*, *convert_after_read=True*, *filter_channel_names=False*, *no_data_loading=False*, *compression=False*, *convert_tables=False*, *metadata=2*)
Bases: *mdfreader.mdf.MdfSkeleton*

mdf file version 3.0 to 3.3 class

fileName
> *str* – file name

MDFVersionNumber
> *int* – mdf file version number

masterChannelList
> *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.

convertAfterRead
> *bool* – flag to convert raw data to physical just after read

filterChannelNames
> *bool* – flag to filter long channel names from its module names separated by '.'

fileMetadata
> *dict* – file metadata with minimum keys: author, organisation, project, subject, comment, time, date

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

_get_channel_data3 (*channelName*)
> Returns channel numpy array

_convert_channel3 (*channelName*)
> converts specific channel from raw to physical data according to CCBlock information

---

_convert_all_channel3()
    Converts all channels from raw data to converted data according to CCBlock information

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

read3(*file_name=None*,   *info=None*,   *multi_processed=False*,   *channel_list=None*,   *convert_after_read=True*, *filter_channel_names=False*, *compression=False*, *metadata=2*)
    Reads mdf 3.x file data and stores it in dict

> **Parameters**
>
> - **file_name** (*str, optional*) – file name
> - **info** (*mdfinfo3.info3 class*) – info3 class containing all MDF Blocks
> - **multi_processed** (*bool*) – flag to activate multiprocessing of channel data conversion
> - **channel_list** (*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
> - **convert_after_read** (*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 .get_channel_data()
> - **filter_channel_names** (*bool, optional*) – flag to filter long channel names from its module names separated by '.'
> - **compression** (*bool, optional*) – flag to activate data compression with blosc
> - **metadata** (*int, optional, default = 2*) – Reading metadata has impact on performance, especially for mdf 4.x using xml. 2: minimal metadata reading (mostly channel blocks) 1: used for noDataLoading 0: all metadata reading

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

> **Parameters file_name** (*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

class mdfreader.mdf3reader.**Record**(*data_group*, *channel_group*)
    Bases: list

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

CGrecordLength
    int – length of record from channel group block information in Byte

recordLength
    int – length of record from channels information in Byte

numberOfRecords
    int – number of records in data block

recordID
    int – recordID corresponding to channel group

---

recordIDnumber
   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

recordToChannelMatching
   dict – helps to identify nested bits in byte

channelNames
   set – channel names to be stored, useful for low memory consumption but slow

hiddenBytes
   Bool, False by default – flag in case of non declared channels in record

byte_aligned
   Bool, True by default – flag for byte aligned record

addChannel(*info*, *channelNumber*)

loadInfo(*info*)

readSortedRecord(*fid*, *pointer*, *channelSet=None*)

readRecordBuf(*buf*, *channelSet=None*)

readRecordBits(*bita*, *channelSet=None*)

add_channel(*info*, *channel_number*)
   add a channel in class

      Parameters

            • **info** (`mdfinfo3.info3 class`) –

            • **channel_number** (`int`) – channel number in mdfinfo3.info3 class

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

      Parameters **info** (`mdfinfo3.info3 class`) –

read_record_bits(*bit_stream*, *channel_set=None*)
   read stream of record bits by bits in case of not aligned or hidden bytes

      Parameters

            • **bit_stream** (`stream`) – stream of bytes read in file

            • **channel_set** (`Set of str, optional`) – list of channel to read

      Returns **rec** – returns dictionary of channel with its corresponding values

      Return type dict

read_record_buf (*buf*, *channel_set=None*)

read stream of record bytes

> **Parameters**
>
> - **buf** (`stream`) – stream of bytes read in file
>
> - **channel_set** (`Set of str, optional`) – list of channel to read
>
> **Returns** **rec** – returns dictionary of channel with its corresponding values
>
> **Return type** dict

read_sorted_record (*fid*, *pointer*, *channel_set=None*)

reads record, only one channel group per data group

> **Parameters**
>
> - **fid** (`float`) – file identifier
>
> - **pointer** – position in file of data block beginning
>
> - **channel_set** (`Set of str, optional`) – list of channel to read
>
> **Returns** **rec** – contains a matrix of raw data in a recarray (attributes corresponding to channel name)
>
> **Return type** numpy recarray

Notes

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

MDFINFO3 MODULE DOCUMENTATION

Measured Data Format blocks parser for version 3.x

Created on Thu Dec 9 12:57:28 2014

Author Aymeric Rateau

4.1 Dependencies

- Python >2.6, >3.2 <http://www.python.org>

- Numpy >1.6 <http://numpy.scipy.org>

mdfreader.mdfinfo3.**PythonVersion**
float – Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

4.2 mdfinfo3

class mdfreader.mdfinfo3.**Info3**(*) -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)*

Bases: dict

clean_dg_info(*dg*)
delete CN,CC and CG blocks related to data group

Parameters dg (*int*) – data group number

fid

fileName

filterChannelNames

list_channels3(*file_name=None*, *fid=None*)
reads data, channel group and channel blocks to list channel names

file_name
str – file name

Returns

Return type list of channel names

read_cg_block (*fid*, *dg*, *minimal=0*)
>   read all CG blocks and relying CN & CC

>   **Parameters**

>   - **fid** (*float*) – file identifier

>   - **dg** (*int*) – data group number

>   - **minimal** (*int*) – 0 will load every metadata 1 will load DG, CG, CN and CC 2 will load only DG

read_info3 (*fid*, *minimal=0*)
>   read all file blocks except data

>   **Parameters**

>   - **fid** (*float*) – file identifier

>   - **minimal** (*int*) – 0 will load every metadata 1 will load DG, CG, CN and CC 2 will load only DG

mdfreader.mdfinfo3.**read_cc_block** (*fid*, *pointer*)
>   channel conversion block reading

mdfreader.mdfinfo3.**read_ce_block** (*fid*, *pointer*)
>   reads source block

mdfreader.mdfinfo3.**read_cg_block** (*fid*, *pointer*)
>   channel block reading

mdfreader.mdfinfo3.**read_cn_block** (*fid*, *pointer*)
>   channel block reading

mdfreader.mdfinfo3.**read_dg_block** (*fid*, *pointer*)
>   data group block reading

mdfreader.mdfinfo3.**read_hd_block** (*fid*, *pointer*, *version=0*)
>   header block reading

mdfreader.mdfinfo3.**read_tx_block** (*fid*, *pointer*)
>   reads text block

MDF4READER MODULE DOCUMENTATION

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

> **Author** Aymeric Rateau

Created on Thu Dec 10 12:57:28 2013

5.1 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

mdfreader.mdf4reader.**PythonVersion**
> *float* – Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

5.2 mdf4reader

class mdfreader.mdf4reader.**Data**(*) -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)*

> Bases: dict

> **add_record**(*record*)
> > Adds a new record in Data class dict.
> >
> > > **Parameters record** (*class*) – channel group definition listing record channel classes

> **fid**

> **load**(*record*, *info*, *name_list=None*, *sorted_flag=True*, *vlsd=False*)
> > Reads data block from record definition
> >
> > > **Parameters**
> > >
> > > - **record** (*class*) – channel group definition listing record channel classes
> > >
> > > - **info** (*class*) – contains blocks

- **name_list** (`list of str, optional`) – list of channel names
- **sorted_flag** (`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
- **vlsd** (`bool`) – indicate a sd block, compressed (DZ) or not (SD)

> **Returns**

> **Return type** numpy recarray of data

pointer_to_data

read (*channel_set*, *info*, *filename*)
> Reads data block

> **Parameters**

> - **channel_set** (`set of str`) – set of channel names
> - **info** (`info object`) – contains blocks structures
> - **filename** – name of file ot read

read_record (*record_id*, *info*, *buf*)
> read record from a buffer

> **Parameters**

> - **record_id** (`int`) – record identifier
> - **info** (`class`) – contains blocks
> - **buf** (`str`) – buffer of data from file to be converted to channel raw data

type

class mdfreader.mdf4reader.**Mdf4** (*file_name=None*, *channel_list=None*, *convert_after_read=True*, *filter_channel_names=False*, *no_data_loading=False*, *compression=False*, *convert_tables=False*, *metadata=2*)
Bases: *mdfreader.mdf.MdfSkeleton*

mdf file reader class from version 4.0 to 4.1.1

fileName
> *str* – file name

MDFVersionNumber
> *int* – mdf file version number

masterChannelList
> *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.

convertAfterRead
> *bool* – flag to convert raw data to physical just after read

filterChannelNames
> *bool* – flag to filter long channel names from its module names separated by '.'

fileMetadata
> *dict* – file metadata with minimum keys : author, organisation, project, subject, comment, time, date

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

_get_channel_data_4 (*channelName*)
> Returns channel numpy array

_convert_channel_data_4 (*channel*, *channel_name*, *convert_tables*, *multiProc=False*, *Q=None*)
> select right conversion and calculates it

_convert_channel_4 (*channelName*)
> converts specific channel from raw to physical data according to CCBlock information

_convert_all_channel_4 ()
> Converts all channels from raw data to converted data according to CCBlock information

write4 (*file_name=None*, *compression=False*)
> writes mdf 4.1 file

apply_invalid_bit (*channel_name*)
> mask data from invalid bit channel if existing

get_channel_name_4 (*name*, *path*)
> returns a list of tuples

apply_invalid_bit (*channel_name*)
> Mask data of channel based on its invalid bit definition if there is

>> **Parameters** **channel_name** (`str`) – Name of channel

get_channel_name4 (*name*, *path*)
> finds mdf channel name from name and path

>> **Parameters**

>> - **name** (`str`) – channel name

>> - **path** (`str`) – source path or name, or channel group name, source name or path

>> **Returns**

>> **Return type** list of tuples (channel_name, (ndg, ncg, ncn))

read4 (*file_name=None*, *info=None*, *multi_processed=False*, *channel_list=None*, *convert_after_read=True*, *compression=False*, *metadata=2*)
> Reads mdf 4.x file data and stores it in dict

>> **Parameters**

>> - **file_name** (`str, optional`) – file name

>> - **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

>> - **multi_processed** (`bool`) – flag to activate multiprocessing of channel data conversion

>> - **channel_list** (`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

>> - **convert_after_read** (`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 .get_channel_data()

- **compression** (*bool, optional*) – flag to activate data compression with blosc

- **metadata** (*int, optional, default = 2*) – Reading metadata has impact on performance, especially for mdf 4.x using xml. 2: minimal metadata reading (mostly channel blocks) 1: used for noDataLoading 0: all metadata reading, including Source Information, Attachment, etc..

write4 (*file_name=None*, *compression=False*)
    Writes simple mdf 4.1 file

        Parameters

- **file_name** (*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

- **compression** (*bool*) – flag to store data compressed

      Notes

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

class mdfreader.mdf4reader.**Record**(*data_group*, *channel_group*)
    Bases: list

    CANOpen

    CGrecordLength

    Flags

    MLSD

    VLSD

    VLSD_CG

    add_channel (*info*, *channel_number*)
        add a channel in class

            Parameters

- **info** (*mdfinfo4.info4 class*) –

- **channel_number** (*int*) – channel number in mdfinfo4.info4 class

    byte_aligned

    channelGroup

    channelNames

    dataGroup

    dataRecordName

    generate_chunks ()
        calculate data split

            Returns

            Return type (n_record_chunk, chunk_size)

    hiddenBytes

initialise_recarray(*info*, *channel_set*, *n_records*, *dtype=None*, *channels_indexes=None*)
> Initialise recarray

>> **Parameters**

>>> - **info** (*info class*) –
>>> - **channel_set** (*set of str, optional*) – set of channel to read
>>> - **n_records** (*int*) – number of records
>>> - **dtype** (*numpy dtype, optional*) –
>>> - **channels_indexes** (*list of int, optional*) –

>> **Returns** **rec** – contains a matrix of raw data in a recarray (attributes corresponding to channel name)

>> **Return type** numpy recarray

invalid_channel

load_info(*info*)
> gathers records related from info class

>> **Parameters** **info** (*mdfinfo4.info4 class*) –

master

numberOfRecords

numpyDataRecordFormat

read_all_channels_sorted_record(*fid*)
> reads all channels from file using numpy fromstring, chunk by chunk

>> **Parameters** **fid** – file identifier

>> **Returns** **rec** – contains a matrix of raw data in a recarray (attributes corresponding to channel name)

>> **Return type** numpy recarray

read_channels_from_bytes(*bit_stream*, *info*, *channel_set=None*, *n_records=None*, *dtype=None*, *channels_indexes=None*)
> reads stream of record bytes using dataRead module if available otherwise bitarray

>> **Parameters**

>>> - **bit_stream** (*stream*) – stream of bytes
>>> - **info** (*info class*) –
>>> - **channel_set** (*set of str, optional*) – set of channel to read
>>> - **n_records** (*int*) – number of records
>>> - **dtype** (*numpy dtype*) –
>>> - **channels_indexes** (*list of int*) –

>> **Returns** **rec** – contains a matrix of raw data in a recarray (attributes corresponding to channel name)

>> **Return type** numpy recarray

read_channels_from_bytes_fallback(*bit_stream*, *info*, *channel_set=None*, *n_records=None*, *dtype=None*, *channels_indexes=None*)
> reads stream of record bytes using bitarray in case no dataRead available

> **Parameters**
>
> - **bit_stream** (*stream*) – stream of bytes
> - **info** (*info class*) –
> - **channel_set** (*set of str, optional*) – set of channel to read
> - **n_records** (*int*) – number of records
> - **dtype** (*numpy dtype*) –
> - **channels_indexes** (*list of int*) –
>
> **Returns rec** – contains a matrix of raw data in a recarray (attributes corresponding to channel name)
>
> **Return type** numpy recarray

read_not_all_channels_sorted_record(*fid*, *info*, *channel_set*)

> reads channels from file listed in channelSet
>
> **Parameters**
>
> - **fid** – file identifier
> - **info** (*info class*) –
> - **channel_set** (*set of str, optional*) – set of channel to read
>
> **Returns rec** – contains a matrix of raw data in a recarray (attributes corresponding to channel name)
>
> **Return type** numpy recarray

read_record_buf(*buf*, *info*, *channel_set=None*)

> read stream of record bytes
>
> **Parameters**
>
> - **buf** (*stream*) – stream of bytes read in file
> - **info** (*class*) – contains blocks structure
> - **channel_set** (*set of str, optional*) – set of channel to read
>
> **Returns rec** – returns dictionary of channel with its corresponding values
>
> **Return type** dict

read_sorted_record(*fid*, *info*, *channel_set=None*)

> reads record, only one channel group per datagroup
>
> **Parameters**
>
> - **fid** – file identifier
> - **info** – info class
> - **channel_set** (*set of str, optional*) – set of channel to read
>
> **Returns rec** – contains a matrix of raw data in a recarray (attributes corresponding to channel name)
>
> **Return type** numpy recarray

Notes

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

recordID

recordIDCFormat

recordIDsize

recordLength

recordToChannelMatching

MDFINFO4 MODULE DOCUMENTATION

Measured Data Format blocks parser for version 4.x

Created on Sun Dec 15 12:57:28 2013

> **Author** Aymeric Rateau

mdfreader.mdfinfo4.**PythonVersion**
> *float* – Python version currently running, needed for compatibility of both python 2.6+ and 3.2+

6.1 mdfinfo4

class mdfreader.mdfinfo4.**ATBlock**(*fid*, *pointer*)
> Bases: dict

> reads Attachment block and saves in class dict

class mdfreader.mdfinfo4.**CABlock**
> Bases: dict

> reads Channel Array block and saves in class dict

> **load**(*byte_offset_base*)

> **read**(*fid*, *pointer*)

> **write**(*fid*)

class mdfreader.mdfinfo4.**CCBlock**
> Bases: dict

> reads Channel Conversion block and saves in class dict

> **read_cc**(*fid*, *pointer*)

class mdfreader.mdfinfo4.**CGBlock**(*fid=None*, *pointer=None*)
> Bases: dict

> reads Channel Group block and saves in class dict

> **read**(*fid*, *pointer*)

> **write**(*fid*)

class mdfreader.mdfinfo4.**CHBlock**(*fid*, *pointer*)
> Bases: dict

> reads Channel Hierarchy block and saves in class dict

class mdfreader.mdfinfo4.**CNBlock**

    Bases: dict

    reads Channel block and saves in class dict

    read_cn(***kargs*)

    write(*fid*)

class mdfreader.mdfinfo4.**CommentBlock**

    Bases: dict

    reads or writes Comment block and saves in class dict

    load(*data*, *md_type*)

    read_cm_at(*fid*, *pointer*)

        reads Comment block from attachment block

            Parameters

                • **fid** – file identifier

                • **pointer** (*int*) – position in file

    read_cm_cc(*fid*, *pointer*)

        reads Comment block from channel conversion block

            Parameters

                • **fid** – file identifier

                • **pointer** (*int*) – position in file

    read_cm_cc_unit(*fid*, *pointer*)

        reads Comment block for channel conversion unit

            Parameters

                • **fid** – file identifier

                • **pointer** (*int*) – position in file

    read_cm_cg(*fid*, *pointer*)

        reads Comment block from channel group block

            Parameters

                • **fid** – file identifier

                • **pointer** (*int*) – position in file

    read_cm_ch(*fid*, *pointer*)

        reads Comment block from file channel hierarchy block

            Parameters

                • **fid** – file identifier

                • **pointer** (*int*) – position in file

    read_cm_cn(*fid*, *pointer*, *minimal=True*)

        reads Comment block from channel block

            Parameters

                • **fid** – file identifier

                • **pointer** (*int*) – position in file

> • **minimal** (*boolean*) – flag to reduce metadata parsing

read_cm_cn_unit (*fid*, *pointer*)
  reads Comment block for channel unit

> **Parameters**
>
>   > • **fid** – file identifier
>   >
>   > • **pointer** (*int*) – position in file

read_cm_dg (*fid*, *pointer*)
  reads Comment block from data group block

> **Parameters**
>
>   > • **fid** – file identifier
>   >
>   > • **pointer** (*int*) – position in file

read_cm_ev (*fid*, *pointer*)
  reads Comment block from event block

> **Parameters**
>
>   > • **fid** – file identifier
>   >
>   > • **pointer** (*int*) – position in file

read_cm_fh (*fid*, *pointer*)
  reads Comment block from file history block

> **Parameters**
>
>   > • **fid** – file identifier
>   >
>   > • **pointer** (*int*) – position in file

read_cm_hd (*fid*, *pointer*)
  reads Comment block from header block

> **Parameters**
>
>   > • **fid** – file identifier
>   >
>   > • **pointer** (*int*) – position in file

read_cm_header (*fid*, *pointer*)
  reads Comment block header

> **Parameters**
>
>   > • **fid** – file identifier
>   >
>   > • **pointer** (*int*) – position in file

read_cm_si (*fid*, *pointer*)
  reads Comment block from source information block

> **Parameters**
>
>   > • **fid** – file identifier
>   >
>   > • **pointer** (*int*) – position in file

read_tx (*fid*, *pointer*)
  reads TX block

> **Parameters**

> - **fid** – file identifier
>
> - **pointer** (*int*) – position in file

read_xml(*fid*)

> reads Comment block xml and objectifies it
>
> > **Parameters** **fid** – file identifier

write(*fid*)

class mdfreader.mdfinfo4.**DGBlock**(*fid=None*, *pointer=None*)

> Bases: dict
>
> reads Data Group block and saves in class dict
>
> **read**(*fid*, *pointer*)
>
> **write**(*fid*)

class mdfreader.mdfinfo4.**DLBlock**

> Bases: dict
>
> reads Data List block
>
> **read**(*fid*, *link_count*)
>
> **write**(*fid*, *chunks*, *position*)

class mdfreader.mdfinfo4.**DTBlock**() -> *new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)*

> Bases: dict
>
> **load**(*record_byte_offset*, *nRecords*, *pointer*)
>
> **write**(*fid*, *data*)

class mdfreader.mdfinfo4.**DZBlock**

> Bases: dict
>
> reads Data List block
>
> **static decompress_data_block**(*block*, *zip_type*, *zip_parameter*, *org_data_length*)
>
> > decompress datablock.
> >
> > **Parameters**
> >
> > - **block** (*bytes*) – raw data compressed
> >
> > - **zip_type** (*int*) – 0 for non transposed, 1 for transposed data
> >
> > - **zip_parameter** (*int*) – first dimension of matrix to be transposed
> >
> > - **org_data_length** (*int*) – uncompressed data length
> >
> > **Returns**
> >
> > **Return type** uncompressed raw data
>
> **read**(*fid*)
>
> **write**(*fid*, *data*, *record_length*)

---

class mdfreader.mdfinfo4.**EVBlock**(*fid*, *pointer*)
    Bases: dict

    reads Event block and saves in class dict

class mdfreader.mdfinfo4.**FHBlock**(*fid=None*, *pointer=None*)
    Bases: dict

    reads File History block and save in class dict

    read(*fid*, *pointer*)

    write(*fid*)

class mdfreader.mdfinfo4.**HDBlock**(*fid=None*)
    Bases: dict

    reads Header block and save in class dict

    read(*fid=None*)

    write(*fid*)

class mdfreader.mdfinfo4.**HLBlock**
    Bases: dict

    reads Header List block

    load(*record_byte_offset*, *n_records*, *position*)

    read(*fid*)

    write(*fid*, *data*)

class mdfreader.mdfinfo4.**IDBlock**(*fid=None*)
    Bases: dict

    reads or writes ID Block

    read(*fid*)
        reads IDBlock

    static write(*fid*)
        Writes IDBlock

class mdfreader.mdfinfo4.**Info4**(*) -> new empty dictionary dict(mapping) -> new dictionary ini-
                             tialized from a mapping object's (key, value) pairs dict(iterable)
                             -> new dictionary initialized as if via: d = {} for k, v in iterable:
                             d[k] = v dict(**kwargs) -> new dictionary initialized with the
                             name=value pairs in the keyword argument list. For example:
                             dict(one=1, two=2)*
    Bases: dict

    clean_dg_info(*dg*)
        delete CN,CC and CG blocks related to data group

            Parameters dg (*int*) – data group number

    fid

    fileName

    list_channels4(*file_name=None*, *fid=None*)
        Read MDF file and extract its complete structure

            Parameters

- **file_name** (`str`) – file name

- **fid** –

    Returns

    Return type list of channel names contained in file

read_cg_block(*fid*, *dg*, *channel_name_list=False*, *minimal=0*)
    reads Channel Group blocks

    Parameters

    - **fid** (`float`) – file identifier

    - **dg** (`int`) – data group number

    - **channel_name_list** (`bool`) – Flag to reads only channel blocks for listChannels4 method

    - **minimal** (`falg`) – to activate minimum content reading for raw data fetching

read_ch_block(*fid*, *pointer*)
    reads channel hierarchy Blocks

    Parameters

    - **fid** (`identifier`) – file identifier

    - **pointer** (`int`) – position of EVBlock in file

    Returns

    Return type channel hierarchy Blocks in a dict

read_cn_block(*fid*, *dg*, *cg*, *channel_name_list=False*, *minimal=0*)
    reads Channel blocks

    Parameters

    - **fid** (`float`) – file identifier

    - **dg** (`int`) – data group number

    - **cg** (`int`) – channel group number in data group

    - **channel_name_list** (`bool`) – Flag to reads only channel blocks for listChannels4 method

    - **minimal** (`flag`) – to activate minimum content reading for raw data fetching

read_composition(*fid*, *dg*, *cg*, *mlsd_channels*)
    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

    - **mlsd_channels** (`list of int`) – channel numbers

    Returns

    Return type MLSDChannels list of appended Maximum Length Sampling Data channels

read_dg_block(*fid*, *channel_name_list=False*, *minimal=0*)
> reads Data Group Blocks

>> **Parameters**

>>> - **fid** (*float*) – file identifier
>>> - **channel_name_list** (*bool*) – Flag to reads only channel blocks for listChannels4 method
>>> - **minimal** (*falg*) – to activate minimum content reading for raw data fetching

static read_ev_block(*fid*, *pointer*)
> reads Events Blocks

>> **Parameters**

>>> - **fid** (*identifier*) – file identifier
>>> - **pointer** (*int*) – position of EVBlock in file

>> **Returns**

>> **Return type** Event Blocks in a dict

read_info(*fid*, *minimal*)
> read all file blocks except data

>> **Parameters**

>>> - **fid** (*identifier*) – file identifier
>>> - **minimal** (*flag*) – to activate minimum content reading for raw data fetching

static read_sr_block(*fid*, *pointer*)
> reads Sample Reduction Blocks

>> **Parameters**

>>> - **fid** (*float*) – file identifier
>>> - **pointer** (*int*) – position of SRBlock in file

>> **Returns**

>> **Return type** Sample Reduction Blocks in a dict

unique_id(*ndg*, *ncg*, *ncn*)
> generate unique id tuples

>> **Parameters**

>>> - **ndg** (*int*) – data group number
>>> - **ncg** (*int*) – channel group number
>>> - **ncn** (*int*) – channel number

>> **Returns tuples** – (channel name, channel source, channel path), (group name, group source, group path)

>> **Return type** (data group number, channel group number, channel number),

zipfile

class mdfreader.mdfinfo4.**SIBlock**
> Bases: dict

> reads Source Information block and saves in class dict

> **read_si** (*fid*, *pointer*)

class mdfreader.mdfinfo4.**SRBlock** (*fid*, *pointer*)
> Bases: dict

> reads Sample Reduction block and saves in class dict

CHANNEL MODULE DOCUMENTATION

Measured Data Format file reader module.

Author Aymeric Rateau

Created on Wed Oct 04 21:13:28 2017

mdfreader.channel.**PythonVersion**
> *float* – Python version currently running, needed for compatibility of both python 2.6+ and 3.4+

mdfreader.channel.**channel**

----

class mdfreader.channel.**Channel3**(*info*, *data_group*, *channel_group*, *channel_number*, *record_id_number*)
> Bases: object
>
> Channel class gathers all about channel structure in a record
>
> **name**
> > *str* – Name of channel
>
> **unit**
> > *str, default empty string* – channel unit
>
> **desc**
> > *str* – channel description
>
> **conversion**
> > *info class* – conversion dictionnary
>
> **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
>
> **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

recAttributeName
   str – channel name compliant to a valid python identifier (recarray attribute)

RecordFormat
   list of str – dtype format used for numpy.core.records functions ((name_title,name),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

bit_masking_needed
   bool, default false – True if bit masking needed after data read

__init__(*info*, *dataGroup*, *channelGroup*, *channelNumber*, *recordIDnumber*)
   constructor

__str__()
   to print class attributes

change_channel_name(*channel_group*)
   rename duplicated channel name within unsorted channel groups

change_channel_name(*channel_group*)
   In case of duplicate channel names within several channel groups for unsorted data, rename channel name

      Parameters **channel_group** (*int*) – channelGroup number

class mdfreader.channel.**Channel4**
   Bases: object

   CANOpen_offset()
      CANopen channel bytes offset

         Returns

         Return type integer, channel bytes offset

   VLSD_CG_Flag

   attachment(*fid*, *info*)
      In case of sync channel attached to channel

         Parameters

               • **fid** (*class*) – file identifier

               • **info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks

         Returns

         Return type ATBlock class from mdfinfo4 module

   bit_count(*info*)
      calculates channel number of bits

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**
>
> **Return type** integer corresponding to channel number of bits

bit_masking_need(*info*)
   Valid if bit masking need

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**
>
> **Return type** boolean True if channel needs bit masking, otherwise False

bit_offset(*info*)
   channel data bit offset in record

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**
>
> **Return type** integer, channel bit offset

byteOffset

c_format(*info*)
   channel data C format

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**
>
> **Return type** string data C format

c_format_structure(*info*)
   channel data C format struct object

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**
>
> **Return type** string data C format struct object

ca_block(*info*)
   Extracts channel CA Block from info4

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**
>
> **Return type** CABlock object from mdfinfo4 module

calc_byte_offset(*info*)
   channel data bytes offset in record (without record id)

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**
>
> **Return type** integer, channel bytes offset

calc_bytes(*info*)
   calculates channel bytes number

> **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
>
> **Returns**

> **Return type** number of bytes integer

change_channel_name(*channel_group*)

> In case of duplicate channel names within several channel groups for unsorted data, rename channel name
>
> > **Parameters channel_group** (*int*) – channelGroup number

channelGroup

channelNumber

channel_sync_type(*info*)

> Extracts channel sync type from info4
>
> > **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
> >
> > **Returns**
> >
> > > - *integer corresponding to channel sync type*
> > > - *0 no sync, normal data*
> > > - *1 time*
> > > - *2 angle*
> > > - *3 distance*
> > > - *4 index*

channel_type(*info*)

> Extracts channel type from info4
>
> > **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
> >
> > **Returns**
> >
> > > - *integer describing channel type*
> > > - *0 normal channel*
> > > - *1 variable length*
> > > - *2 master channel*
> > > - *3 virtual master channel*
> > > - *4 sync channel*
> > > - *5 max length data*
> > > - *6 virtual data channel*

cn_block(*info*)

> channel block
>
> > **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
> >
> > **Returns**
> >
> > **Return type** CNBlock class from mdfinfo4 module

conversion(*info*)

> channel conversion CCBlock
>
> > **Parameters info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks
> >
> > **Returns**
> >
> > **Return type** *CCBlock*

data(*info*)
   returns data block pointer for VLSD, MLD or sync channels

dataGroup

data_format(*info*)
   channel numpy.core.records data format

   > **Parameters** **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

   > **Returns**

   > **Return type** string data format

desc(*info*)
   channel description

   > **Parameters** **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

   > **Returns**

   > **Return type** channel description string

has_invalid_bit(*info*)

invalid_bit(*info*)
   extracts from info4 the channels valid bits positions

   > **Parameters** **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

   > **Returns**

   > **Return type** channel valid bit position

is_ca_block(*info*)

isnumeric(*info*)
   check this is numeric channel from data type

   > **Parameters** **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

   > **Returns**

   > **Return type** boolean, true if numeric channel, otherwise false

little_endian(*info*)
   check if channel is little endian

   > **Parameters** **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

   > **Returns**

   > **Return type** boolean

nBytes

name

native_data_format(*info*)

numpy_format(*info*)
   channel numpy.core.records data format

   > **Parameters** **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

   > **Returns** endian, dataType

   > **Return type** string data format

pos_bit_beg(*info*)

    channel data bit starting position in record

        Parameters info (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks

        Returns

        Return type integer, channel bit starting position

pos_bit_end(*info*)

    channel data bit ending position in record

        Parameters info (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks

        Returns

        Return type integer, channel bit ending position

pos_byte_beg(*info*)

    channel data bytes starting position in record

        Parameters info (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks

        Returns

        Return type integer, channel bytes starting position

pos_byte_end(*info*)

    channel data bytes ending position in record

        Parameters info (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks

        Returns

        Return type integer, channel bytes ending position

record_attribute_name()

    clean up channel name from unauthorised characters

        Returns

        Return type channel name compliant to python attributes names (for recarray)

record_id_size(*info*)

    Extracts record id size from info4

        Parameters info (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks

        Returns

             • *integer describing record id size*

             • *0 no record id used*

             • *1 uint8*

             • *2 uint16*

             • *4 uint32*

             • *8 uint64*

set(*info*, *data_group*, *channel_group*, *channel_number*)

    channel initialisation

        Parameters

             • **info** (*mdfinfo4.info4 class*) –

- **data_group** (*int*) – data group number in mdfinfo4.info4 class

- **channel_group** (*int*) – channel group number in mdfinfo4.info4 class

- **channel_number** (*int*) – channel number in mdfinfo4.info4 class

set_CANOpen(*info*, *data_group*, *channel_group*, *channel_number*, *name*)
    CANOpen channel intialisation

    Parameters

- **info** (*mdfinfo4.info4 class*) –

- **data_group** (*int*) – data group number in mdfinfo4.info4 class

- **channel_group** (*int*) – channel group number in mdfinfo4.info4 class

- **channel_number** (*int*) – channel number in mdfinfo4.info4 class

- **name** (*str*) – name of channel. Should be in ('ms', 'day', 'days', 'hour', 'month', 'minute', 'year')

set_invalid_bytes(*info*, *data_group*, *channel_group*, *channel_number*)
    invalid_bytes channel initialisation

    Parameters

- **info** (*mdfinfo4.info4 class*) –

- **data_group** (*int*) – data group number in mdfinfo4.info4 class

- **channel_group** (*int*) – channel group number in mdfinfo4.info4 class

- **channel_number** (*int*) – channel number in mdfinfo4.info4 class

signal_data_type(*info*, *byte_aligned=True*)
    extract signal data type from info4 class

    Parameters

- **info** (*mdfinfo4.info4 class*) – info4 class containing all MDF Blocks

- **byte_aligned** (*bool*) – flag activated if channel is part of a record byte aligned

    Returns

- *integer corresponding to channel data type*

- *0 unsigned integer little endian*

- *1 unsigned integer big endian*

- *2 signed integer little endian*

- *3 signed integer big endian*

- *4 float little endian*

- *5 float big endian*

- *6 string latin*

- *7 string utf-8*

- *9 string utf-16*

- *10 byte array*

- *11 mime sample*

- *12 mime stream*

- *13 CANopen date*

- *14 CANopen time*

type

unit(*info*)

    channel unit

        Parameters **info** (`mdfinfo4.info4 class`) – info4 class containing all MDF Blocks

        Returns

        Return type channel unit string

mdfreader.channel.**array_format4**(*signal_data_type*, *number_of_bytes*)

    function returning numpy style string from channel data type and number of bits

    Parameters

        - **signal_data_type** (*int*) – channel data type according to specification

        - **number_of_bytes** (*int*) – number of bytes taken by channel data in a record

    Returns **endian, data_type** – numpy dtype format used by numpy.core.records to read channel raw
        data

    Return type str

mdfreader.channel.**data_type_format4**(*signal_data_type*, *number_of_bytes*)

    function returning C format string from channel data type and number of bits

    Parameters

        - **signal_data_type** (*int*) – channel data type according to specification

        - **number_of_bytes** (*int*) – number of bytes taken by channel data in a record

    Returns **data_type** – C format used by fread to read channel raw data

    Return type str

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

Symbols

A

B

C

W

Z