mdfreader Documentation
Release 1.4

Aymeric Rateau

Oct 22, 2017

Contents:

MDF MODULE DOCUMENTATION

mdf_skeleton module describing basic mdf structure and methods

Created on Thu Sept 24 2015

1.1 Platform and python version

With Unix and Windows for python 2.6+ and 3.2+

> **Author** Aymeric Rateau

1.2 Dependencies

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

1.3 mdf module

class `mdf.`**`compressed_data`**
> class to represent compressed data by blosc

> ### **Methods**

> **`compression`**(*a*)
> > data compression method

> > > **Parameters a** : numpy array

> > > > data to be compresses

> **`decompression`**()
> > data decompression

class `mdf.`**`mdf_skeleton`**(*fileName=None*, *channelList=None*, *convertAfterRead=True*, *filterChannel-Names=False*, *noDataLoading=False*, *compression=False*)
> Bases: `dict`

Methods

MDFVersionNumber

add_channel (*dataGroup*, *channel_name*, *data*, *master_channel*, *master_type=1*, *unit=''*, *description=''*, *conversion=None*, *info=None*, *compression=False*)

   adds channel to mdf dict.

> **Parameters dataGroup** : int
>
>> dataGroup number. Is appended to master name for non unique channel names
>
> **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

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

convertAfterRead

---

convert_tables

copy()
> copy a mdf class

fid

fileName

file_metadata

filterChannelNames

getChannel(*channelName*)
> Extract channel dict from mdf structure

>> **Parameters channelName** : str

>>> channel name

>> **Returns** channel dictionnary containing data, description, unit, etc.

getChannelConversion(*channelName*)
> Extract channel conversion dict from mdf structure

>> **Parameters channelName** : str

>>> channel name

>> **Returns** channel conversion dict

getChannelDesc(*channelName*)
> Extract channel description information from mdf structure

>> **Parameters channelName** : str

>>> channel name

>> **Returns** channel description string

getChannelMaster(*channelName*)
> Extract channel master name from mdf structure

>> **Parameters channelName** : str

>>> channel name

>> **Returns** channel master name string

getChannelMasterType(*channelName*)
> Extract channel master type information from mdf structure

>> **Parameters channelName** : str

>>> channel name

>> **Returns** channel mater type integer

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

>> **Parameters channelName** : str

>>> channel name

>> **Returns** str

>>> unit string description

info

masterChannelList

multiProc

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

> **Parameters channel_name** : str
>
> > channel name
>
> **Returns** value of mdf dict key=channel_name

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

> **Parameters channelName** : str
>
> > channel name
>
> **Returns** removed value from dict

rename_channel(*channelName*, *newname*)
   Modifies name of channel

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

setChannelAttachment(*channelName*, *attachment*)
   Modifies channel attachment

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

setChannelConversion(*channelName*, *conversion*)
   Modifies conversion dict of channel

> **Parameters channelName** : str
>
> > channel name
>
> **conversion** : dict
>
> > conversion dictionnary

setChannelData(*channelName*, *data*, *compression=False*)
   Modifies data of channel

> **Parameters channelName** : str
>
> > channel name
>
> **data** : numpy array
>
> > channel data
>
> **compression** : bool or str

> trigger for data compression

setChannelDesc(*channelName*, *desc*)
    Modifies description of channel

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

setChannelMaster(*channelName*, *master*)
    Modifies channel master name

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

setChannelMasterType(*channelName*, *masterType*)
    Modifies master channel type

>   **Parameters channelName** : str
>
>   channel name
>
>   **masterType** : int
>
>   master channel type

setChannelUnit(*channelName*, *unit*)
    Modifies unit of channel

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

zipfile

MDFREADER MODULE DOCUMENTATION

Measured Data Format file reader main module

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
- 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

2.3 Attributes

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

2.4 mdfreader module

class `mdfreader.`**`mdf`**(*fileName=None, channelList=None, convertAfterRead=True, filterChannel-Names=False, noDataLoading=False, compression=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
# 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(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.write() # writes mdf file
# drops all the channels except the one in argument
>>> yop.keepChannels({'channel1','channel2','channel3'})
>>> yop.getChannelData('channelName') # returns channel numpy array
```

Attributes

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.
file_metadata	(dict) file metadata with minimum keys : author, organisation, project, subject, comment, time, date

Methods

read(fileName = None, multiProc = False, channelList=None, convertAfterRead=True, filterChannelNames=False, noDataLoading=False, compression=False)	reads mdf file version 3.x and 4.x
write(fileName=None)	writes simple mdf 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

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

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).

exportToCSV (*filename=None*, *sampling=None*)

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. 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

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 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

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

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*, *noDataLoading=False*, *compression=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 '.'

noDataLoading : 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

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=None*, *masterChannel=None*)

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

> **Parameters samplingTime** : 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****
>
> > **masterChannel** : 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(*fileName=None*)

Writes simple mdf file, same format as originally read, default is 4.x

> **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*, *fid=None*, *minimal=False*)

Bases: dict

Methods

fid

fileName

filterChannelNames

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

mdfversion

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

        Parameters fileName : str, optional

            file name. If not input, uses fileName attribute

           fid : file identifier, optional

zipfile

MDF3READER MODULE DOCUMENTATION

Measured Data Format file reader module for version 3.x

3.1 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

3.2 Dependencies

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

3.3 Attributes

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

3.4 mdf3reader module

class `mdf3reader.`**`Channel`**(*info*, *dataGroup*, *channelGroup*, *channelNumber*, *recordIDnumber*)
    Channel class gathers all about channel structure in a record

Attributes

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
recAttribute-Name	(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

Methods

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

changeChannelName (*channelGroup*)

> In case of duplicate channel names within several channel groups for unsorted data, rename channel name

> > **Parameters channelGroup** : int

> > > channelGroup bumber

class `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.

Attributes

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

Methods

addRecord(record)	Adds a new record in DATA class dict
read(channelSet)	Reads data block
loadSorted(record, nameList=None)	Reads sorted data block from record definition
loadUnSorted(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

loadSorted(*record*, *nameList=None*)

    Reads sorted data block from record definition

        Parameters record class

            channel group definition listing record channel classes

        channelSet : set of str, optional

            list of channel names

        Returns numpy recarray of data

loadUnSorted(*nameList=None*)

    Reads unsorted data block from record definition

        Parameters record class

            channel group definition listing record channel classes

        channelSet : set of str, optional

            list of channel names

        Returns numpy recarray of data

read(*channelSet*)

    Reads data block

        Parameters channelSet : set of str, optional

            list of channel names

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*, *channelList=None*, *convertAfterRead=True*, *filterChannelNames=False*, *noDataLoading=False*, *compression=False*)
: Bases: *mdf.mdf_skeleton*

mdf file version 3.0 to 3.3 class

Attributes

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 '.'
file_metadata	(dict) file metadata with minimum keys: author, organisation, project, subject, comment, time, date

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

read3 (*fileName=None*, *info=None*, *multiProc=False*, *channelList=None*, *convertAfterRead=True*, *filterChannelNames=False*, *compression=False*)
   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()
   >
   > **compression** : bool, optional
   >
   >> falg to activate data compression with blosc

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.**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 Channel classes,** it is representing a channel group

Attributes

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
recordToChannelMatch-ing	(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

Methods

addChannel(info, channelNumber)	
loadInfo(info)	
readSortedRecord(fid, pointer, channelSet=None)	
readRecordBuf(buf, channelSet=None)	
readRecordBits(bita, channelSet=None)	
changeChannelName(channelName)	

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

> > **Parameters** **info** : mdfinfo3.info3 class

> > > **channelNumber** : int

> > > > channel number in mdfinfo3.info3 class

changeChannelName(*channelName*)
> In case of duplicate channel names within several channel groups for unsorted data, rename channel name

Parameters channelName : str

channelName to be modified to avoid duplicates in unsorted channel groups

loadInfo(*info*)

gathers records related from info class

Parameters info : mdfinfo3.info3 class

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

read stream of record bits by bits in case of not aligned or hidden bytes

Parameters buf : stream

stream of bytes read in file

channelSet : Set of str, optional

list of channel to read

Returns rec : dict

returns dictionary of channel with its corresponding values

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

read stream of record bytes

Parameters buf : stream

stream of bytes read in file

channelSet : Set of str, optional

list of channel to read

Returns rec : dict

returns dictionary of channel with its corresponding values

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

reads record, only one channel group per datagroup

Parameters fid : float

file identifier

pointer

position in file of data block beginning

channelSet : Set 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 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.

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

4.1 Platform and python version

With Unix and Windows for python 2.6+ and 3.2+

> **Author** Aymeric Rateau

4.2 Dependencies

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

4.3 Attributes

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

4.4 mdfinfo3 module

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

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

cleanDGinfo(*dg*)

> delete CN,CC and CG blocks related to data group

> > **Parameters dg** : int
> >
> > > data group number

fid

fileName

filterChannelNames

listChannels3(*fileName=None*, *fid=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

readCGBlock(*fid*, *dg*, *minimal=False*)

> read all CG blocks and relying CN & CC

> > **Parameters fid** : float
> >
> > > file identifier
> >
> > **dg** : int
> >
> > > datagroup number
> >
> > **channelSet** : set
> >
> > > set of channel names to read
> >
> > **minimal** : bool
> >
> > > flag to request minimal info loading. Later loaded by data read datagroup by datagroup

readinfo3 (*fid*, *minimal=False*)
read all file blocks except data

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

MDF4READER MODULE DOCUMENTATION

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

5.1 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

5.2 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

5.3 Attributes

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

5.4 mdf4reader module

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

Methods

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

> **Parameters** record class

channel group definition listing record channel classes

fid

load(*record*, *info*, *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

pointerTodata

read(*channelSet*, *info*, *zip=None*)
    Reads data block

> **Parameters channelSet** : set of str
>
> > set of channel names
> >
> > **zip** : bool, optional
> >
> > > flag to track if data block is compressed

readRecord(*recordID*, *info*, *buf*, *channelSet=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
> >
> > **channelSet** : set of str
> >
> > setof channel names to be read

type

mdf4reader.**DATABlock**(*record*, *info*, *parent_block*, *channelSet=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
> >
> > **channelSet** : set of str, optional
> >
> > defines set 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.**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.**decompress_datablock**(*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** uncompressed 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

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*, *channelList=None*, *convertAfterRead=True*, *filterChannel-Names=False*, *noDataLoading=False*, *compression=False*)

> Bases: *mdf.mdf_skeleton*
>
> mdf file reader class from version 4.0 to 4.1.1

Attributes

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 '.'
file_metadata	(dict) file metadata with minimum keys : author, organisation, project, subject, comment, time, date

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

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

> **Parameters** **fileName** : str, optional
>
>> file name
>
> **info** : mdfinfo4.info4 class
>
>> info4 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()
>
> **compression** : bool, optional
>
>> falg to activate data compression with blosc

write4 (*fileName=None*)
Writes simple mdf 4.1 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

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

Methods

CANOpen

CGrecordLength

Flags

MLSD

VLSD

VLSD_CG

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

　　　　Parameters info : mdfinfo4.info4 class

　　　　　　channelNumber : int

　　　　　　　　channel number in mdfinfo4.info4 class

byte_aligned

channelGroup

channelNames

dataGroup

dataRecordName

hiddenBytes

invalid_channel

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

　　　　Parameters info : mdfinfo4.info4 class

master

numberOfRecords

numpyDataRecordFormat

readBitarray(*bita*, *info*, *channelSet=None*)
　　reads stream of record bytes using bitarray module needed for not byte aligned data

　　　　Parameters bita : stream

　　　　　　　　stream of bytes

　　　　　　channelSet : set of str, optional

　　　　　　　　set of channel to read

　　　　Returns rec : numpy recarray

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

readRecordBuf(*buf*, *info*, *channelSet=None*)
     read stream of record bytes

> **Parameters buf** : stream
>
>> stream of bytes read in file
>>
>> **channelSet** : set of str, optional
>>
>> set of channel to read
>
> **Returns rec** : dict
>
>> returns dictionary of channel with its corresponding values

readSortedRecord(*fid*, *pointer*, *info*, *channelSet=None*)
     reads record, only one channel group per datagroup

> **Parameters fid** : float
>
>> file identifier
>>
>> **pointer**
>>
>> position in file of data block beginning
>>
>> **channelSet** : set of str, optional
>>
>> set of channel to read
>
> **Returns rec** : numpy recarray
>
>> contains a matrix of raw data in a recarray (attributes corresponding to channel name)

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

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

6.1 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

6.2 Dependencies

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

6.3 Attributes

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

6.4 mdfinfo4 module

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

   reads Attachment block and saves in class dict

   ### Methods

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

   reads Channel Array block and saves in class dict

Methods

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

 reads Channel Conversion block and saves in class dict

 Methods

 read(*fid*, *pointer*)

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

 reads Channel Group block and saves in class dict

 Methods

 read(*fid*, *pointer*)

 write(*fid*, *cg_cycle_count*, *cg_data_bytes*)

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

 reads Channel Hierarchy block and saves in class dict

 Methods

class mdfinfo4.**CNBlock**
 Bases: dict

 reads Channel block and saves in class dict

 Methods

 read(*fid*, *pointer*)

 write(*fid*)

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

 Methods

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

   Parameters xml_tree : xml tree from xml.etree.ElementTree

    field : str

   Returns field value in xml tree

read (*fid*, *pointer*, *MDType=None*)
> reads Comment block and saves in class dict

Notes

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

write (*fid*, *data*, *MDType*)

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

> reads Data Group block and saves in class dict

Methods

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

> **write** (*fid*)

class mdfinfo4.**DLBlock** (*fid*, *link_count*)
> Bases: dict

> reads Data List block

Methods

class mdfinfo4.**DZBlock** (*fid*)
> Bases: dict

> reads Data List block

Methods

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

> reads Event block and saves in class dict

Methods

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

> reads File History block and save in class dict

Methods

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

> **write** (*fid*)

class `mdfinfo4.HDBlock`(*fid=None*, *pointer=64*)
> Bases: `dict`

> reads Header block and save in class dict

> ### Methods

> > **read**(*fid=None*, *pointer=64*)

> > **write**(*fid*)

class `mdfinfo4.HLBlock`(*fid*)
> Bases: `dict`

> reads Header List block

> ### Methods

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

> reads or writes ID Block

> ### Methods

> > **read**(*fid*)
> > > reads IDBlock

> > **write**(*fid*)
> > > Writes IDBlock

class `mdfinfo4.SIBlock`
> Bases: `dict`

> reads Source Information block and saves in class dict

> ### Methods

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

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

> 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*, *minimal=False*)
> Bases: `dict`

Methods

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

   Parameters dg : int

    data group number

fid

fileName

listChannels4(*fileName=None*, *fid=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*, *minimal=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

    minimal: falg

     to activate minimum content reading for raw data fetching

readCNBlock(*fid*, *dg*, *cg*, *channelNameList=False*, *minimal=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

minimal: falg

to activate minimum content reading for raw data fetching

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*, *minimal=False*)
reads Data Group Blocks

Parameters fid : float

file identifier

channelNameList : bool

Flag to reads only channel blocks for listChannels4 method

minimal: falg

to activate minimum content reading for raw data fetching

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*, *minimal*)
read all file blocks except data

Parameters fid : float

file identifier

minimal: falg

to activate minimum content reading for raw data fetching

`zipfile`

CHANNEL MODULE DOCUMENTATION

Measured Data Format file reader module.

7.1 Platform and python version

With Unix and Windows for python 2.7 and 3.4+

> **Author** Aymeric Rateau

Created on Wed Oct 04 21:13:28 2017

7.2 Dependencies

- Python >2.6, >3.4 <http://www.python.org>
- Numpy >1.6 <http://numpy.scipy.org>

7.3 Attributes

PythonVersion [float] Python version currently running, needed for compatibility of both python 2.6+ and 3.4+

7.4 channel module

channel.**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

class channel.**channel4**
> Bases: object

Methods

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

>   **Parameters info** : mdfinfo4.info4 class
>
>>   info4 class containing all MDF Blocks
>
>   **Returns** CABlock object from mdfinfo4 module

CANOpenOffset (*info*)
    CANopen channel bytes offset

>   **Parameters info** : mdfinfo4.info4 class
>
>>   info4 class containing all MDF Blocks
>
>   **Returns** integer, channel bytes offset

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

>   **Parameters info** : mdfinfo4.info4 class
>
>>   info4 class containing all MDF Blocks
>
>   **Returns** string data C format struct object

CNBlock (*info*)
    channel block

>   **Parameters info** : mdfinfo4.info4 class
>
>>   info4 class containing all MDF Blocks
>
>   **Returns** CNBlock class from mdfinfo4 module

Format (*info*)
    channel data C format

>   **Parameters info** : mdfinfo4.info4 class
>
>>   info4 class containing all MDF Blocks
>
>   **Returns** string data C format

RecordFormat (*info*)
    recarray dtype string

>   **Parameters info** : mdfinfo4.info4 class
>
>>   info4 class containing all MDF Blocks
>
>   **Returns** recarray dtype

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** ATBlock class from mdfinfo4 module

bitCount (*info*)

    calculates channel number of bits

> **Parameters info** : mdfinfo4.info4 class
>
> > info4 class containing all MDF Blocks
>
> **Returns** integer corresponding to channel number of bits

bitOffset (*info*)

    channel data bit offset in record

> **Parameters info** : mdfinfo4.info4 class
>
> > info4 class containing all MDF Blocks
>
> **Returns** integer, channel bit offset

byteOffset (*info*)

    channel data bytes offset in record (without record id)

> **Parameters info** : mdfinfo4.info4 class
>
> > info4 class containing all MDF Blocks
>
> **Returns** integer, channel bytes offset

channelGroup

channelNumber

channelSyncType (*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

channelType (*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

conversion(*info*)
 channel conversion CCBlock

> **Parameters info** : mdfinfo4.info4 class
>
>> info4 class containing all MDF Blocks
>
> **Returns** CCBlock

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

> **Parameters info** : mdfinfo4.info4 class
>
>> info4 class containing all MDF Blocks
>
> **Returns** string data format

dataGroup

desc(*info*)
 channel description

> **Parameters info** : mdfinfo4.info4 class
>
>> info4 class containing all MDF Blocks
>
> **Returns** channel description string

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

> **Parameters info** : mdfinfo4.info4 class
>
>> info4 class containing all MDF Blocks
>
> **Returns** dict of channels valid bits positions

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

> **Parameters info** : mdfinfo4.info4 class
>
>> info4 class containing all MDF Blocks
>
> **Returns** boolean

nBytes(*info*)
 calculates channel bytes number

> **Parameters info** : mdfinfo4.info4 class
>
>> info4 class containing all MDF Blocks
>
> **Returns** number of bytes integer

name

nativeRecordFormat(*info*)
 channel numpy.core.records data format without endian (< or >)

> **Parameters info** : mdfinfo4.info4 class
>
>> info4 class containing all MDF Blocks

> **Returns** string data format

posBitBeg(*info*)
  channel data bit starting position in record

> > **Parameters info** : mdfinfo4.info4 class
> >
> > > info4 class containing all MDF Blocks
> >
> > **Returns** integer, channel bit starting position

posBitEnd(*info*)
  channel data bit ending position in record

> > **Parameters info** : mdfinfo4.info4 class
> >
> > > info4 class containing all MDF Blocks
> >
> > **Returns** integer, channel bit ending position

posByteBeg(*info*)
  channel data bytes starting position in record

> > **Parameters info** : mdfinfo4.info4 class
> >
> > > info4 class containing all MDF Blocks
> >
> > **Returns** integer, channel bytes starting position

posByteEnd(*info*)
  channel data bytes ending position in record

> > **Parameters info** : mdfinfo4.info4 class
> >
> > > info4 class containing all MDF Blocks
> >
> > **Returns** integer, channel bytes ending position

recAttributeName(*info*)
  clean up channel name from unauthorised characters

> > **Parameters info** : mdfinfo4.info4 class
> >
> > > info4 class containing all MDF Blocks
> >
> > **Returns** channel name compliant to python attributes names (for recarray)

recordIDsize(*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*, *dataGroup*, *channelGroup*, *channelNumber*)
  channel initialisation

> **Parameters info** : mdfinfo4.info4 class
>
>> **dataGroup** : int
>>
>>> data group number in mdfinfo4.info4 class
>>
>> **channelGroup** : int
>>
>>> channel group number in mdfinfo4.info4 class
>>
>> **channelNumber** : int
>>
>>> channel number in mdfinfo4.info4 class
>>
>> **recordIDsize** : int
>>
>>> size of record ID in Bytes

setCANOpen(*info*, *dataGroup*, *channelGroup*, *channelNumber*, *name*)
    CANOpen channel intialisation

> **Parameters info** : mdfinfo4.info4 class
>
>> **dataGroup** : int
>>
>>> data group number in mdfinfo4.info4 class
>>
>> **channelGroup** : int
>>
>>> channel group number in mdfinfo4.info4 class
>>
>> **channelNumber** : int
>>
>>> channel number in mdfinfo4.info4 class
>>
>> **recordIDsize** : int
>>
>>> size of record ID in Bytes
>>
>> **name** : str
>>
>>> name of channel. Should be in ('ms', 'day', 'days', 'hour', 'month', 'minute', 'year')

setInvalidBytes(*info*, *dataGroup*, *channelGroup*, *channelNumber*)
    invalid_bytes channel initialisation

> **Parameters info** : mdfinfo4.info4 class
>
>> **dataGroup** : int
>>
>>> data group number in mdfinfo4.info4 class
>>
>> **channelGroup** : int
>>
>>> channel group number in mdfinfo4.info4 class
>>
>> **channelNumber** : int
>>
>>> channel number in mdfinfo4.info4 class
>>
>> **recordIDsize** : int
>>
>>> size of record ID in Bytes
>>
>> **byte_aligned** : Bool
>>
>>> Flag for byte alignement

signalDataType(*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** channel unit string

validity_channel (*info*, *invalid_bytes*)

> extract channel validity bits
>
> > **Parameters info** : mdfinfo4.info4 class
> >
> > > **invalid_bytes** : bytes
> > >
> > > bytes from where to extract validity bit array

channel.**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

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

c

channel, 45

m

mdf, 3
mdf3reader, 17
mdf4reader, 29
mdfinfo3, 25
mdfinfo4, 37
mdfreader, 9

A

add_channel() (mdf.mdf_skeleton method), 4
add_metadata() (mdf.mdf_skeleton method), 4
addChannel() (mdf3reader.record method), 22
addChannel() (mdf4reader.record method), 34
addRecord() (mdf3reader.DATA method), 19
addRecord() (mdf4reader.DATA method), 29
allPlot() (mdfreader.mdf method), 11
append_field() (in module mdf4reader), 31
arrayformat4() (in module channel), 45
ATBlock (class in mdfinfo4), 37
attachment() (channel.channel4 method), 46

B

bitCount() (channel.channel4 method), 47
bitOffset() (channel.channel4 method), 47
blockformats3() (mdfinfo3.info3 static method), 25
byte_aligned (mdf4reader.record attribute), 34
byteOffset() (channel.channel4 method), 47

C

CABlock (class in mdfinfo4), 37
CABlock() (channel.channel4 method), 46
CANOpen (mdf4reader.record attribute), 34
CANOpenOffset() (channel.channel4 method), 46
CCBlock (class in mdfinfo4), 38
CFormat() (channel.channel4 method), 46
CGBlock (class in mdfinfo4), 38
CGrecordLength (mdf4reader.record attribute), 34
change_field_name() (in module mdf4reader), 31
changeChannelName() (mdf3reader.Channel method), 18
changeChannelName() (mdf3reader.record method), 22
Channel (class in mdf3reader), 17
channel (module), 45
channel4 (class in channel), 45
channelGroup (channel.channel4 attribute), 47
channelGroup (mdf4reader.record attribute), 34
channelNames (mdf4reader.record attribute), 34
channelNumber (channel.channel4 attribute), 47
channelSyncType() (channel.channel4 method), 47
channelType() (channel.channel4 method), 47
CHBlock (class in mdfinfo4), 38

cleanDGinfo() (mdfinfo3.info3 method), 26
cleanDGinfo() (mdfinfo4.info4 method), 41
CNBlock (class in mdfinfo4), 38
CNBlock() (channel.channel4 method), 46
CommentBlock (class in mdfinfo4), 38
compressed_data (class in mdf), 3
compression() (mdf.compressed_data method), 3
conversion() (channel.channel4 method), 48
convert_tables (mdf.mdf_skeleton attribute), 4
convertAfterRead (mdf.mdf_skeleton attribute), 4
convertAllChannel() (mdfreader.mdf method), 11
convertToPandas() (mdfreader.mdf method), 12
copy() (mdf.mdf_skeleton method), 5
copy() (mdfreader.mdf method), 12
cut() (mdfreader.mdf method), 12

D

DATA (class in mdf3reader), 18
DATA (class in mdf4reader), 29
DATABlock() (in module mdf4reader), 30
dataFormat() (channel.channel4 method), 48
dataGroup (channel.channel4 attribute), 48
dataGroup (mdf4reader.record attribute), 34
dataRecordName (mdf4reader.record attribute), 34
datatypeformat4() (in module channel), 51
decompress_datablock() (in module mdf4reader), 31
decompression() (mdf.compressed_data method), 3
desc() (channel.channel4 method), 48
DGBlock (class in mdfinfo4), 39
DLBlock (class in mdfinfo4), 39
DZBlock (class in mdfinfo4), 39

E

elementTreeToDict() (in module mdfinfo4), 40
equalizeStringLength() (in module mdf4reader), 31
EVBlock (class in mdfinfo4), 39
expConv() (in module mdf3reader), 19
exportToCSV() (mdfreader.mdf method), 12
exportToExcel() (mdfreader.mdf method), 12
exportToHDF5() (mdfreader.mdf method), 12
exportToMatlab() (mdfreader.mdf method), 13
exportToNetCDF() (mdfreader.mdf method), 13