mdfreader Documentation

Release 2.7.5

Aymeric Rateau

Jan 19, 2018

CONTENTS

1 mdf module documentation **3**
1.1 Platform and python version . . . 3
1.2 Dependencies . . . 3
1.3 mdf module . . . 3

2 mdfreader module documentation **9**
2.1 Platform and python version . . . 9
2.2 Dependencies . . . 9
2.3 Attributes . . . 9
2.4 mdfreader module . . . 10

3 mdf3reader module documentation **19**
3.1 Platform and python version . . . 19
3.2 Dependencies . . . 19
3.3 Attributes . . . 19
3.4 mdf3reader module . . . 19

4 mdfinfo3 module documentation **27**
4.1 Platform and python version . . . 27
4.2 Dependencies . . . 27
4.3 Attributes . . . 27
4.4 mdfinfo3 module . . . 27

5 mdf4reader module documentation **31**
5.1 Platform and python version . . . 31
5.2 Dependencies . . . 31
5.3 Attributes . . . 31
5.4 mdf4reader module . . . 31

6 mdfinfo4 module documentation **43**
6.1 Platform and python version . . . 43
6.2 Dependencies . . . 43
6.3 Attributes . . . 43
6.4 mdfinfo4 module . . . 43

7 channel module documentation **59**
7.1 Platform and python version . . . 59
7.2 Dependencies . . . 59
7.3 Attributes . . . 59
7.4 channel module . . . 59

8 Indices and tables 69

Python Module Index 71

Index 73

Contents:

CHAPTER

ONE

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 `mdfreader.mdf.`**`compressed_data`**

Methods

`compression`(a)	data compression method
`decompression`()	data decompression

`compression`(*a*)

data compression method

Parameters **a** : numpy array

data to be compresses

`decompression`()

data decompression

class `mdfreader.mdf.`**`mdf_skeleton`**(*fileName=None*, *channelList=None*, *convertAfterRead=True*, *filterChannelNames=False*, *noDataLoading=False*, *compression=False*)

Bases: `dict`

Methods

`add_channel`(dataGroup, channel_name, data, ...)	adds channel to mdf dict.
`add_metadata`([author, organisation, ...])	adds basic metadata to mdf class
`clear`(() -> None. Remove all items from D.)	
`copy`()	copy a mdf class
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`getChannel`(channelName)	Extract channel dict from mdf structure
`getChannelConversion`(channelName)	Extract channel conversion dict from mdf structure
`getChannelDesc`(channelName)	Extract channel description information from mdf structure
`getChannelMaster`(channelName)	Extract channel master name from mdf structure
`getChannelMasterType`(channelName)	Extract channel master type information from mdf structure
`getChannelUnit`(channelName)	Returns channel unit string
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`remove_channel`(channel_name)	removes channel from mdf dict.
`remove_channel_conversion`(channelName)	removes conversion key from mdf channel dict.
`rename_channel`(channelName, newname)	Modifies name of channel
`setChannelAttachment`(channelName, attachment)	Modifies channel attachment
`setChannelConversion`(channelName, conversion)	Modifies conversion dict of channel
`setChannelData`(channelName, data[, compression])	Modifies data of channel
`setChannelDesc`(channelName, desc)	Modifies description of channel
`setChannelMaster`(channelName, master)	Modifies channel master name
`setChannelMasterType`(channelName, masterType)	Modifies master channel type
`setChannelUnit`(channelName, unit)	Modifies unit of channel
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	

Continued on next page

Table 1.2 – continued from previous page

`viewvalues(...)`

`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`

CHAPTER

TWO

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.mdfreader.`**`mdf`**(*fileName=None*, *channelList=None*, *convertAfterRead=True*, *filterChannelNames=False*, *noDataLoading=False*, *compression=False*)

Bases: `mdfreader.mdf3reader.mdf3`, `mdfreader.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()

make a shallow 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*, *compression=None*, *compression_opts=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.

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

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*, *raw_data=False*)
Return channel numpy array

Parameters **channelName** : str

channel name

raw_data: bool

flag to return non converted data

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.mdfreader.`**`mdfinfo`**(*fileName=None*, *filterChannelNames=False*, *fid=None*, *minimal=0*)

Bases: `dict`

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`listChannels`([fileName])	Read MDF file blocks and returns a list of contained channels
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`readinfo`([fileName, fid, minimal])	Reads MDF file and extracts its complete structure
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

`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=0*)
Reads MDF file and extracts its complete structure

Parameters **fileName** : 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`

CHAPTER

THREE

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 `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.

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*, *filename*)
Reads data block

Parameters **channelSet** : set of str, optional

list of channel names

filename : str

name of file

`mdfreader.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

`mdfreader.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

`mdfreader.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

`mdfreader.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 `mdfreader.mdf3reader.`**`mdf3`**(*fileName=None*, *channelList=None*, *convertAfterRead=True*, *filterChannelNames=False*, *noDataLoading=False*, *compression=False*)

Bases: *`mdfreader.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

mdfreader.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

`mdfreader.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 `mdfreader.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
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

Methods

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

`addChannel`(*info*, *channelNumber*)

add a channel in class

Parameters **info** : mdfinfo3.info3 class

channelNumber : int

channel number in mdfinfo3.info3 class

`loadInfo`(*info*)

gathers records related from info class

Parameters **info** : mdfinfo3.info3 class

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.

mdfreader.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

`mdfreader.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

`mdfreader.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

CHAPTER

FOUR

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 `mdfreader.mdfinfo3.`**`info3`**(*fileName=None*, *fid=None*, *filterChannelNames=False*, *minimal=0*)

Bases: `dict`

Methods

`cleanDGinfo`(dg)	delete CN,CC and CG blocks related to data group
`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	

Continued on next page

Table 4.1 – continued from previous page

`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`listChannels3`([fileName, fid])	reads data, channel group and channel blocks to list channel names
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`readCGBlock`(fid, dg[, minimal])	read all CG blocks and relying CN & CC
`readinfo3`(fid[, minimal])	read all file blocks except data
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

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

readCGBlock(*fid*, *dg*, *minimal=0*)

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 : int

0 will load every metadata 1 will load DG, CG, CN and CC 2 will load only DG

`readinfo3` (*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_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

CHAPTER

FIVE

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 `mdfreader.mdf4reader.`**`DATA`**(*fid*, *pointer*)

Bases: `dict`

Methods

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

Continued on next page

Table 5.1 – continued from previous page

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`load`(record, info[, nameList, sortedFlag, vlsd])	Reads data block from record definition
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`read`(channelSet, info, filename)	Reads data block
`readRecord`(recordID, info, buf[, channelSet])	read record from a buffer
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

addRecord(*record*)

Adds a new record in DATA class dict.

Parameters record class

channel group definition listing record channel classes

fid

load(*record*, *info*, *nameList=None*, *sortedFlag=True*, *vlsd=False*)

Reads data block from record definition

Parameters record class

channel group definition listing record channel classes

info class

contains blocks

nameList : list of str, optional

list of channel names

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

vlsd : bool

indicate a sd block, compressed (DZ) or not (SD)

Returns numpy recarray of data

`pointerTodata`

`read` (*channelSet*, *info*, *filename*)
Reads data block

Parameters **channelSet** : set of str

set of channel names

info : info object

contains blocks structures

filename

name of file ot read

`readRecord` (*recordID*, *info*, *buf*, *channelSet=None*)
read record from a buffer

Parameters **recordID** : int

record identifier

info class

contains blocks

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`

`mdfreader.mdf4reader.`**`DATABlock`** (*record*, *info*, *parent_block*, *channelSet=None*, *nrecords=None*, *sortedFlag=True*, *vlsd=False*)
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

nrecords: int, optional

number of records to read

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

vlsd : bool

indicate a sd block, compressed (DZ) or not (SD)

Returns a recarray containing the channels data

Notes

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

`mdfreader.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

`mdfreader.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

`mdfreader.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

`mdfreader.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 `mdfreader.mdf4reader.`**`mdf4`**(*fileName=None*, *channelList=None*, *convertAfterRead=True*, *filterChannelNames=False*, *noDataLoading=False*, *compression=False*)

Bases: *mdfreader.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

`mdfreader.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

`mdfreader.mdf4reader.`**`readUnsorted`**(*record*, *info*, *parent_block*, *channelSet=None*)

`mdfreader.mdf4reader.`**`read_sdblock`**(*signal_data_type*, *sdblock*, *sdblock_length*)
Reads vlsd channel from its SD Block bytes

Parameters **signal_data_type** : int

sdblock : bytes

SD Block bytes

sdblock_length: int

SD Block data length (header not included)

Returns array

class `mdfreader.mdf4reader.`**`record`**(*dataGroup*, *channelGroup*)
Bases: `list`

Methods

addChannel(info, channelNumber)	add a channel in class
append	L.append(object) – append object to end
count(...)	
extend	L.extend(iterable) – extend list by appending elements from the iterable
generate_chunks()	Initialise recarray
index((value, [start, ...)	Raises ValueError if the value is not present.
initialise_recarray(info, channelSet, nrecords)	Initialise recarray
insert	L.insert(index, object) – insert object before index
loadInfo(info)	gathers records related from info class

Continued on next page

Table 5.2 – continued from previous page

pop(...)	Raises IndexError if list is empty or index is out of range.
readRecordBuf(buf, info[, channelSet])	read stream of record bytes
readSortedRecord(fid, info[, channelSet])	reads record, only one channel group per datagroup
read_all_channels_sorted_record(fid)	reads all channels from file using numpy fromstring, chunk by chunk
read_channels_from_bytes(bita, info[, ...])	reads stream of record bytes using dataRead module if available otherwise bitarray
read_channels_from_bytes_fallback(bita, info)	reads stream of record bytes using bitarray in case no dataRead available
read_not_all_channels_sorted_record(fid, ...)	reads channels from file listed in channelSet
remove	L.remove(value) – remove first occurrence of value.
reverse	L.reverse() – reverse *IN PLACE*
sort	L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*;

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

generate_chunks()
Initialise recarray

Returns (nrecord_chunk, chunk_size)

hiddenBytes

initialise_recarray(*info*, *channelSet*, *nrecords*, *dtype=None*, *channels_indexes=None*)
Initialise recarray

Parameters **info: info class**

channelSet : set of str, optional

set of channel to read

nrecords: int

number of records

dtype: numpy dtype, optional

channels_indexes: list of int, optional

Returns rec : numpy recarray

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

`invalid_channel`

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

Parameters info : mdfinfo4.info4 class

`master`

`numberOfRecords`

`numpyDataRecordFormat`

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

Parameters buf : stream

stream of bytes read in file

info class

contains blocks structure

channelSet : set of str, optional

set of channel to read

Returns rec : dict

returns dictionary of channel with its corresponding values

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

Parameters fid :

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.

`read_all_channels_sorted_record`(*fid*)

reads all channels from file using numpy fromstring, chunk by chunk

Parameters **fid :**

file identifier

Returns **rec** : numpy recarray

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

`read_channels_from_bytes`(*bita*, *info*, *channelSet=None*, *nrecords=None*, *dtype=None*, *channels_indexes=None*)

reads stream of record bytes using dataRead module if available otherwise bitarray

Parameters **bita** : stream

stream of bytes

info: info class

channelSet : set of str, optional

set of channel to read

nrecords: int

number of records

dtype: numpy dtype

channels_indexes: list of int

Returns **rec** : numpy recarray

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

`read_channels_from_bytes_fallback`(*bita*, *info*, *channelSet=None*, *nrecords=None*, *dtype=None*, *channels_indexes=None*)

reads stream of record bytes using bitarray in case no dataRead available

Parameters **bita** : stream

stream of bytes

info: info class

channelSet : set of str, optional

set of channel to read

nrecords: int

number of records

dtype: numpy dtype

channels_indexes: list of int

Returns **rec** : numpy recarray

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

read_not_all_channels_sorted_record(*fid*, *info*, *channelSet*)
reads channels from file listed in channelSet

Parameters **fid :**

file identifier

info: info class

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)

recordID

recordIDCFormat

recordIDsize

recordLength

recordToChannelMatching

mdfreader.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

mdfreader.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

mdfreader.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

mdfreader.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

`mdfreader.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

`mdfreader.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

`mdfreader.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

CHAPTER

SIX

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 `mdfreader.mdfinfo4.`**`ATBlock`** (*fid*, *pointer*)

Bases: `dict`

reads Attachment block and saves in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	

Continued on next page

Table 6.1 – continued from previous page

has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	

class mdfreader.mdfinfo4.**CABlock** (*fid*, *pointer*)

Bases: dict

reads Channel Array block and saves in class dict

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	

class mdfreader.mdfinfo4.**CCBlock**

Bases: dict

reads Channel Conversion block and saves in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`read`(fid, pointer)	
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

`read` (*fid*, *pointer*)

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

Bases: `dict`

reads Channel Group block and saves in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`read`(fid, pointer)	
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]

Continued on next page

Table 6.4 – continued from previous page

`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	
`write`(fid)	

`read` (*fid*, *pointer*)

`write` (*fid*)

class `mdfreader.mdfinfo4.`**`CHBlock`** (*fid*, *pointer*)

Bases: `dict`

reads Channel Hierarchy block and saves in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

class `mdfreader.mdfinfo4.`**`CNBlock`**

Bases: `dict`

reads Channel block and saves in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.

Continued on next page

Table 6.6 – continued from previous page

get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
read(**kargs)	
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	
write(fid)	

read(***kargs*)

write(*fid*)

class mdfreader.mdfinfo4.**CommentBlock**

Bases: dict

reads or writes Comment block and saves in class dict

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
load(data, MDType)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
read(**kargs)	reads Comment block and saves in class dict
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	

Continued on next page

Table 6.7 – continued from previous page

viewitems(...)	
viewkeys(...)	
viewvalues(...)	
write(fid)	

load (*data*, *MDType*)

read (***kargs*)

reads Comment block and saves in class dict Parameters ——— fid: file identifier pointer: int

position in file

MDType: str describes metadata type, ('CN', 'unit', 'FH', 'SI', 'HD', 'CC')

Notes

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

write (*fid*)

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

Bases: dict

reads Data Group block and saves in class dict

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
read(fid, pointer)	
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	
write(fid)	

read (*fid*, *pointer*)

write (*fid*)

class mdfreader.mdfinfo4.**DLBlock** (*fid*, *link_count*)

Bases: dict

reads Data List block

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	

class mdfreader.mdfinfo4.**DZBlock** (*fid*)

Bases: dict

reads Data List block

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised

Continued on next page

Table 6.10 – continued from previous page

`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

class `mdfreader.mdfinfo4.`**`EVBlock`** (*fid*, *pointer*)

Bases: `dict`

reads Event block and saves in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

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

Bases: `dict`

reads File History block and save in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.

Continued on next page

Table 6.12 – continued from previous page

get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
read(fid, pointer)	
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	
write(fid)	

read (*fid*, *pointer*)

write (*fid*)

class mdfreader.mdfinfo4.**HDBlock** (*fid=None*, *pointer=64*)

Bases: dict

reads Header block and save in class dict

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
read([fid, pointer])	
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	

Continued on next page

Table 6.13 – continued from previous page

viewkeys(...)	
viewvalues(...)	
write(fid)	

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

write (*fid*)

class mdfreader.mdfinfo4.**HLBlock** (*fid*)

Bases: dict

reads Header List block

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	

class mdfreader.mdfinfo4.**IDBlock** (*fid=None*)

Bases: dict

reads or writes ID Block

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	

Continued on next page

Table 6.15 – continued from previous page

items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
read(fid)	reads IDBlock
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	
write(fid)	Writes IDBlock

read (*fid*)
reads IDBlock

write (*fid*)
Writes IDBlock

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

reads Source Information block and saves in class dict

Methods

clear(() -> None. Remove all items from D.)	
copy(() -> a shallow copy of D)	
fromkeys(...)	v defaults to None.
get((k[,d]) -> D[k] if k in D, ...)	
has_key((k) -> True if D has a key k, else False)	
items(() -> list of D's (key, value) pairs, ...)	
iteritems(() -> an iterator over the (key, ...)	
iterkeys(() -> an iterator over the keys of D)	
itervalues(...)	
keys(() -> list of D's keys)	
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
read(fid, pointer)	
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	

Continued on next page

Table 6.16 – continued from previous page

`viewkeys`(...)	
`viewvalues`(...)	

`read`(*fid*, *pointer*)

class `mdfreader.mdfinfo4.`**`SRBlock`**(*fid*, *pointer*)

Bases: `dict`

reads Sample Reduction block and saves in class dict

Methods

`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	
`itervalues`(...)	
`keys`(() -> list of D's keys)	
`pop`((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
`popitem`(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
`setdefault`((k[,d]) -> D.get(k,d), ...)	
`update`(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
`values`(() -> list of D's values)	
`viewitems`(...)	
`viewkeys`(...)	
`viewvalues`(...)	

class `mdfreader.mdfinfo4.`**`info4`**(*fileName=None*, *fid=None*, *minimal=0*)

Bases: `dict`

Methods

`cleanDGinfo`(dg)	delete CN,CC and CG blocks related to data group
`clear`(() -> None. Remove all items from D.)	
`copy`(() -> a shallow copy of D)	
`fromkeys`(...)	v defaults to None.
`get`((k[,d]) -> D[k] if k in D, ...)	
`has_key`((k) -> True if D has a key k, else False)	
`items`(() -> list of D's (key, value) pairs, ...)	
`iteritems`(() -> an iterator over the (key, ...)	
`iterkeys`(() -> an iterator over the keys of D)	

Continued on next page

Table 6.18 – continued from previous page

itervalues(...)	
keys(() -> list of D's keys)	
listChannels4([fileName, fid])	Read MDF file and extract its complete structure
pop((k[,d]) -> v, ...)	If key is not found, d is returned if given, otherwise KeyError is raised
popitem(() -> (k, v), ...)	2-tuple; but raise KeyError if D is empty.
readATBlock(selfself, fid, pointer)	reads Attachment blocks
readCGBlock(fid, dg[, channelNameList, minimal])	reads Channel Group blocks
readCNBlock(fid, dg, cg[, channelNameList, ...])	reads Channel blocks
readComposition(fid, dg, cg, MLSDChannels[, ...])	check for composition of channels, arrays or structures
readDGBlock(fid[, channelNameList, minimal])	reads Data Group Blocks
readSRBlock(fid, pointer)	reads Sample Reduction Blocks
readinfo(fid, minimal)	read all file blocks except data
setdefault((k[,d]) -> D.get(k,d), ...)	
update(([E, ...)	If E present and has a .keys() method, does: for k in E: D[k] = E[k]
values(() -> list of D's values)	
viewitems(...)	
viewkeys(...)	
viewvalues(...)	

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=0*)
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=0*)
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=0*)
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`

CHAPTER

SEVEN

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

class `mdfreader.channel.`**`Channel3`**(*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

`mdfreader.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 **endian, dataType** : str

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

class `mdfreader.channel.`**`channel4`**
Bases: `object`

Methods

CABlock(info)	Extracts channel CA Block from info4
CANOpenOffset(info)	CANopen channel bytes offset
CFormat(info)	channel data C format struct object
CNBlock(info)	channel block
Format(info)	channel data C format
attachment(fid, info)	In case of sync channel attached to channel
bitCount(info)	calculates channel number of bits
bitOffset(info)	channel data bit offset in record
byteOffset(info)	channel data bytes offset in record (without record id)
changeChannelName(channelGroup)	In case of duplicate channel names within several channel groups
channelSyncType(info)	Extracts channel sync type from info4
channelType(info)	Extracts channel type from info4
conversion(info)	channel conversion CCBlock
data(info)	returns data block pointer for VLSD, MLD or sync channels
dataFormat(info)	channel numpy.core.records data format
desc(info)	channel description
invalid_bit(info)	extrzcts from info4 the channels valid bits positions
isCABlock(info)	
little_endian(info)	check if channel is little endian
nBytes(info)	calculates channel bytes number
nativedataFormat(info)	
numpy_format(info)	channel numpy.core.records data format
posBitBeg(info)	channel data bit starting position in record
posBitEnd(info)	channel data bit ending position in record
posByteBeg(info)	channel data bytes starting position in record
posByteEnd(info)	channel data bytes ending position in record
recAttributeName(info)	clean up channel name from unauthorised characters
recordIDsize(info)	Extracts record id size from info4
set(info, dataGroup, channelGroup, channelNumber)	channel initialisation
setCANOpen(info, dataGroup, channelGroup, ...)	CANOpen channel intialisation
setInvalidBytes(info, dataGroup, ...)	invalid_bytes channel initialisation
signalDataType(info[, byte_aligned])	extract signal data type from info4 class
unit(info)	channel unit
validity_channel(info, invalid_bytes)	extract channel validity bits

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

`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

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

Parameters **channelGroup** : int

channelGroup bumber

`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

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

`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

`isCABlock`(*info*)

`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`

`nativedataFormat`(*info*)

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

Parameters **info** : mdfinfo4.info4 class

info4 class containing all MDF Blocks

Returns **endian, dataType** : 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

mdfreader.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

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

m

mdfreader.channel, 59
mdfreader.mdf, 3
mdfreader.mdf3reader, 19
mdfreader.mdf4reader, 31
mdfreader.mdfinfo3, 27
mdfreader.mdfinfo4, 43
mdfreader.mdfreader, 9

A

add_channel() (mdfreader.mdf.mdf_skeleton method), 5
add_metadata() (mdfreader.mdf.mdf_skeleton method), 5
addChannel() (mdfreader.mdf3reader.record method), 23
addChannel() (mdfreader.mdf4reader.record method), 37
addRecord() (mdfreader.mdf3reader.DATA method), 20
addRecord() (mdfreader.mdf4reader.DATA method), 32
allPlot() (mdfreader.mdfreader.mdf method), 11
arrayformat4() (in module mdfreader.channel), 60
ATBlock (class in mdfreader.mdfinfo4), 43
attachment() (mdfreader.channel.channel4 method), 62

B

bitCount() (mdfreader.channel.channel4 method), 62
bitOffset() (mdfreader.channel.channel4 method), 62
byte_aligned (mdfreader.mdf4reader.record attribute), 37
byteOffset() (mdfreader.channel.channel4 method), 62

C

CABlock (class in mdfreader.mdfinfo4), 44
CABlock() (mdfreader.channel.channel4 method), 61
CANOpen (mdfreader.mdf4reader.record attribute), 37
CANOpenOffset() (mdfreader.channel.channel4 method), 61
CCBlock (class in mdfreader.mdfinfo4), 44
CFormat() (mdfreader.channel.channel4 method), 61
CGBlock (class in mdfreader.mdfinfo4), 45
CGrecordLength (mdfreader.mdf4reader.record attribute), 37
changeChannelName() (mdfreader.channel.Channel3 method), 60
changeChannelName() (mdfreader.channel.channel4 method), 62
Channel3 (class in mdfreader.channel), 59
channel4 (class in mdfreader.channel), 60
channelGroup (mdfreader.channel.channel4 attribute), 63
channelGroup (mdfreader.mdf4reader.record attribute), 37
channelNames (mdfreader.mdf4reader.record attribute), 37
channelNumber (mdfreader.channel.channel4 attribute), 63
channelSyncType() (mdfreader.channel.channel4 method), 63
channelType() (mdfreader.channel.channel4 method), 63
CHBlock (class in mdfreader.mdfinfo4), 46
cleanDGinfo() (mdfreader.mdfinfo3.info3 method), 28
cleanDGinfo() (mdfreader.mdfinfo4.info4 method), 55
CNBlock (class in mdfreader.mdfinfo4), 46
CNBlock() (mdfreader.channel.channel4 method), 62
CommentBlock (class in mdfreader.mdfinfo4), 47
compressed_data (class in mdfreader.mdf), 3
compression() (mdfreader.mdf.compressed_data method), 3
conversion() (mdfreader.channel.channel4 method), 63
convert_tables (mdfreader.mdf.mdf_skeleton attribute), 6
convertAfterRead (mdfreader.mdf.mdf_skeleton attribute), 5
convertAllChannel() (mdfreader.mdfreader.mdf method), 11
convertToPandas() (mdfreader.mdfreader.mdf method), 12
copy() (mdfreader.mdf.mdf_skeleton method), 6
copy() (mdfreader.mdfreader.mdf method), 12
cut() (mdfreader.mdfreader.mdf method), 12

D

DATA (class in mdfreader.mdf3reader), 19
DATA (class in mdfreader.mdf4reader), 31
data() (mdfreader.channel.channel4 method), 63
DATABlock() (in module mdfreader.mdf4reader), 33
dataFormat() (mdfreader.channel.channel4 method), 63
dataGroup (mdfreader.channel.channel4 attribute), 64
dataGroup (mdfreader.mdf4reader.record attribute), 37
dataRecordName (mdfreader.mdf4reader.record attribute), 37
datatypeformat4() (in module mdfreader.channel), 67
decompress_datablock() (in module mdfreader.mdf4reader), 34
decompression() (mdfreader.mdf.compressed_data method), 3
desc() (mdfreader.channel.channel4 method), 64
DGBlock (class in mdfreader.mdfinfo4), 48
DLBlock (class in mdfreader.mdfinfo4), 49

DZBlock (class in mdfreader.mdfinfo4), 49

E

equalizeStringLength() (in module mdfreader.mdf4reader), 34
EVBlock (class in mdfreader.mdfinfo4), 50
expConv() (in module mdfreader.mdf3reader), 20
exportToCSV() (mdfreader.mdfreader.mdf method), 12
exportToExcel() (mdfreader.mdfreader.mdf method), 12
exportToHDF5() (mdfreader.mdfreader.mdf method), 12
exportToMatlab() (mdfreader.mdfreader.mdf method), 13
exportToNetCDF() (mdfreader.mdfreader.mdf method), 13
exportToXlsx() (mdfreader.mdfreader.mdf method), 13

F

FHBlock (class in mdfreader.mdfinfo4), 50
fid (mdfreader.mdf.mdf_skeleton attribute), 6
fid (mdfreader.mdf4reader.DATA attribute), 32
fid (mdfreader.mdfinfo3.info3 attribute), 28
fid (mdfreader.mdfinfo4.info4 attribute), 55
fid (mdfreader.mdfreader.mdfinfo attribute), 16
file_metadata (mdfreader.mdf.mdf_skeleton attribute), 6
fileName (mdfreader.mdf.mdf_skeleton attribute), 6
fileName (mdfreader.mdfinfo3.info3 attribute), 28
fileName (mdfreader.mdfinfo4.info4 attribute), 55
fileName (mdfreader.mdfreader.mdfinfo attribute), 16
filterChannelNames (mdfreader.mdf.mdf_skeleton attribute), 6
filterChannelNames (mdfreader.mdfinfo3.info3 attribute), 28
filterChannelNames (mdfreader.mdfreader.mdfinfo attribute), 16
Flags (mdfreader.mdf4reader.record attribute), 37
Format() (mdfreader.channel.channel4 method), 62
formulaConv() (in module mdfreader.mdf3reader), 20
formulaConv() (in module mdfreader.mdf4reader), 34

G

generate_chunks() (mdfreader.mdf4reader.record method), 37
getChannel() (mdfreader.mdf.mdf_skeleton method), 6
getChannelConversion() (mdfreader.mdf.mdf_skeleton method), 6
getChannelData() (mdfreader.mdfreader.mdf method), 14
getChannelDesc() (mdfreader.mdf.mdf_skeleton method), 6
getChannelMaster() (mdfreader.mdf.mdf_skeleton method), 6
getChannelMasterType() (mdfreader.mdf.mdf_skeleton method), 6
getChannelUnit() (mdfreader.mdf.mdf_skeleton method), 6

H

HDBlock (class in mdfreader.mdfinfo4), 51
hiddenBytes (mdfreader.mdf4reader.record attribute), 37
HLBlock (class in mdfreader.mdfinfo4), 52

I

IDBlock (class in mdfreader.mdfinfo4), 52
info (mdfreader.mdf.mdf_skeleton attribute), 7
info3 (class in mdfreader.mdfinfo3), 27
info4 (class in mdfreader.mdfinfo4), 54
initialise_recarray() (mdfreader.mdf4reader.record method), 37
invalid_bit() (mdfreader.channel.channel4 method), 64
invalid_channel (mdfreader.mdf4reader.record attribute), 38
isCABlock() (mdfreader.channel.channel4 method), 64

K

keepChannels() (mdfreader.mdfreader.mdf method), 14

L

linearConv() (in module mdfreader.mdf3reader), 21
linearConv() (in module mdfreader.mdf4reader), 34
listChannels() (mdfreader.mdfreader.mdfinfo method), 16
listChannels3() (mdfreader.mdfinfo3.info3 method), 28
listChannels4() (mdfreader.mdfinfo4.info4 method), 55
little_endian() (mdfreader.channel.channel4 method), 64
load() (mdfreader.mdf4reader.DATA method), 32
load() (mdfreader.mdfinfo4.CommentBlock method), 48
loadInfo() (mdfreader.mdf3reader.record method), 23
loadInfo() (mdfreader.mdf4reader.record method), 38
loadSorted() (mdfreader.mdf3reader.DATA method), 20
loadUnSorted() (mdfreader.mdf3reader.DATA method), 20
logConv() (in module mdfreader.mdf3reader), 21

M

master (mdfreader.mdf4reader.record attribute), 38
masterChannelList (mdfreader.mdf.mdf_skeleton attribute), 7
mdf (class in mdfreader.mdfreader), 10
mdf3 (class in mdfreader.mdf3reader), 21
mdf4 (class in mdfreader.mdf4reader), 34
mdf_skeleton (class in mdfreader.mdf), 3
mdfinfo (class in mdfreader.mdfreader), 16
mdfreader.channel (module), 59
mdfreader.mdf (module), 3
mdfreader.mdf3reader (module), 19
mdfreader.mdf4reader (module), 31
mdfreader.mdfinfo3 (module), 27
mdfreader.mdfinfo4 (module), 43
mdfreader.mdfreader (module), 9
mdfversion (mdfreader.mdfreader.mdfinfo attribute), 16

MDFVersionNumber (mdfreader.mdf.mdf_skeleton attribute), 5
mergeMdf() (mdfreader.mdfreader.mdf method), 14
MLSD (mdfreader.mdf4reader.record attribute), 37
multiProc (mdfreader.mdf.mdf_skeleton attribute), 7

N

name (mdfreader.channel.channel4 attribute), 64
nativedataFormat() (mdfreader.channel.channel4 method), 64
nBytes() (mdfreader.channel.channel4 method), 64
numberOfRecords (mdfreader.mdf4reader.record attribute), 38
numpy_format() (mdfreader.channel.channel4 method), 64
numpyDataRecordFormat (mdfreader.mdf4reader.record attribute), 38

P

plot() (mdfreader.mdfreader.mdf method), 14
pointerTodata (mdfreader.mdf4reader.DATA attribute), 33
polyConv() (in module mdfreader.mdf3reader), 22
posBitBeg() (mdfreader.channel.channel4 method), 64
posBitEnd() (mdfreader.channel.channel4 method), 64
posByteBeg() (mdfreader.channel.channel4 method), 65
posByteEnd() (mdfreader.channel.channel4 method), 65

R

rationalConv() (in module mdfreader.mdf3reader), 23
rationalConv() (in module mdfreader.mdf4reader), 36
read() (mdfreader.mdf3reader.DATA method), 20
read() (mdfreader.mdf4reader.DATA method), 33
read() (mdfreader.mdfinfo4.CCBlock method), 45
read() (mdfreader.mdfinfo4.CGBlock method), 46
read() (mdfreader.mdfinfo4.CNBlock method), 47
read() (mdfreader.mdfinfo4.CommentBlock method), 48
read() (mdfreader.mdfinfo4.DGBlock method), 48
read() (mdfreader.mdfinfo4.FHBlock method), 51
read() (mdfreader.mdfinfo4.HDBlock method), 52
read() (mdfreader.mdfinfo4.IDBlock method), 53
read() (mdfreader.mdfinfo4.SIBlock method), 54
read() (mdfreader.mdfreader.mdf method), 14
read3() (mdfreader.mdf3reader.mdf3 method), 22
read4() (mdfreader.mdf4reader.mdf4 method), 35
read_all_channels_sorted_record() (mdfreader.mdf4reader.record method), 39
read_cc_block() (in module mdfreader.mdfinfo3), 29
read_cg_block() (in module mdfreader.mdfinfo3), 29
read_channels_from_bytes() (mdfreader.mdf4reader.record method), 39
read_channels_from_bytes_fallback() (mdfreader.mdf4reader.record method), 39
read_cn_block() (in module mdfreader.mdfinfo3), 29
read_dg_block() (in module mdfreader.mdfinfo3), 29
read_hd_block() (in module mdfreader.mdfinfo3), 29
read_not_all_channels_sorted_record() (mdfreader.mdf4reader.record method), 39
read_sdblock() (in module mdfreader.mdf4reader), 36
read_tx_block() (in module mdfreader.mdfinfo3), 29
readATBlock() (mdfreader.mdfinfo4.info4 method), 55
readCGBlock() (mdfreader.mdfinfo3.info3 method), 28
readCGBlock() (mdfreader.mdfinfo4.info4 method), 55
readCNBlock() (mdfreader.mdfinfo4.info4 method), 56
readComposition() (mdfreader.mdfinfo4.info4 method), 56
readDGBlock() (mdfreader.mdfinfo4.info4 method), 56
readinfo() (mdfreader.mdfinfo4.info4 method), 57
readinfo() (mdfreader.mdfreader.mdfinfo method), 16
readinfo3() (mdfreader.mdfinfo3.info3 method), 29
readRecord() (mdfreader.mdf4reader.DATA method), 33
readRecordBits() (mdfreader.mdf3reader.record method), 24
readRecordBuf() (mdfreader.mdf3reader.record method), 24
readRecordBuf() (mdfreader.mdf4reader.record method), 38
readSortedRecord() (mdfreader.mdf3reader.record method), 24
readSortedRecord() (mdfreader.mdf4reader.record method), 38
readSRBlock() (mdfreader.mdfinfo4.info4 method), 57
readUnsorted() (in module mdfreader.mdf4reader), 36
recAttributeName() (mdfreader.channel.channel4 method), 65
record (class in mdfreader.mdf3reader), 23
record (class in mdfreader.mdf4reader), 36
recordID (mdfreader.mdf4reader.record attribute), 40
recordIDCFormat (mdfreader.mdf4reader.record attribute), 40
recordIDsize (mdfreader.mdf4reader.record attribute), 40
recordIDsize() (mdfreader.channel.channel4 method), 65
recordLength (mdfreader.mdf4reader.record attribute), 40
recordToChannelMatching (mdfreader.mdf4reader.record attribute), 40
remove_channel() (mdfreader.mdf.mdf_skeleton method), 7
remove_channel_conversion() (mdfreader.mdf.mdf_skeleton method), 7
rename_channel() (mdfreader.mdf.mdf_skeleton method), 7
resample() (mdfreader.mdfreader.mdf method), 15

S

set() (mdfreader.channel.channel4 method), 65
setCANOpen() (mdfreader.channel.channel4 method), 66
setChannelAttachment() (mdfreader.mdf.mdf_skeleton method), 7

setChannelConversion() (mdfreader.mdf.mdf_skeleton method), 7
setChannelData() (mdfreader.mdf.mdf_skeleton method), 7
setChannelDesc() (mdfreader.mdf.mdf_skeleton method), 8
setChannelMaster() (mdfreader.mdf.mdf_skeleton method), 8
setChannelMasterType() (mdfreader.mdf.mdf_skeleton method), 8
setChannelUnit() (mdfreader.mdf.mdf_skeleton method), 8
setInvalidBytes() (mdfreader.channel.channel4 method), 66
SIBlock (class in mdfreader.mdfinfo4), 53
signalDataType() (mdfreader.channel.channel4 method), 66
SRBlock (class in mdfreader.mdfinfo4), 54

T

tabConv() (in module mdfreader.mdf3reader), 24
tabInterpConv() (in module mdfreader.mdf3reader), 24
textRangeTableConv() (in module mdfreader.mdf3reader), 25
textToTextConv() (in module mdfreader.mdf4reader), 40
textToValueConv() (in module mdfreader.mdf4reader), 40
type (mdfreader.channel.channel4 attribute), 67
type (mdfreader.mdf4reader.DATA attribute), 33

U

unit() (mdfreader.channel.channel4 method), 67

V

validity_channel() (mdfreader.channel.channel4 method), 67
valueRangeToTextConv() (in module mdfreader.mdf4reader), 40
valueRangeToValueTableConv() (in module mdfreader.mdf4reader), 40
valueToTextConv() (in module mdfreader.mdf4reader), 41
valueToValueTableWInterpConv() (in module mdfreader.mdf4reader), 41
valueToValueTableWOInterpConv() (in module mdfreader.mdf4reader), 41
VLSD (mdfreader.mdf4reader.record attribute), 37
VLSD_CG (mdfreader.mdf4reader.record attribute), 37
VLSD_CG_Flag (mdfreader.channel.channel4 attribute), 62

W

write() (mdfreader.mdfinfo4.CGBlock method), 46
write() (mdfreader.mdfinfo4.CNBlock method), 47
write() (mdfreader.mdfinfo4.CommentBlock method), 48
write() (mdfreader.mdfinfo4.DGBlock method), 48
write() (mdfreader.mdfinfo4.FHBlock method), 51
write() (mdfreader.mdfinfo4.HDBlock method), 52
write() (mdfreader.mdfinfo4.IDBlock method), 53
write() (mdfreader.mdfreader.mdf method), 15
write3() (mdfreader.mdf3reader.mdf3 method), 22
write4() (mdfreader.mdf4reader.mdf4 method), 36

Z

zipfile (mdfreader.mdf.mdf_skeleton attribute), 8
zipfile (mdfreader.mdfinfo4.info4 attribute), 57
zipfile (mdfreader.mdfreader.mdfinfo attribute), 17