
asammdf Documentation

Release 2.8.1

Daniel Hrisca

Dec 18, 2017

Contents

1	Project goals	3
2	Features	5
3	Major features not implemented (yet)	7
4	Dependencies	9
5	Installation	11
6	API	13
7	Benchmarks	61
8	Indices and tables	71

asammdf is a fast parser/editor for ASAM (Association for Standardisation of Automation and Measuring Systems) MDF (Measurement Data Format) files.

asammdf supports MDF versions 2 (.dat), 3 (.mdf) and 4 (.mf4).

asammdf works on Python 2.7, and Python ≥ 3.4 (Travis CI tests done with Python 2.7 and Python ≥ 3.5)

CHAPTER 1

Project goals

The main goals for this library are:

- to be faster than the other Python based mdf libraries
- to have clean and easy to understand code base

CHAPTER 2

Features

- create new mdf files from scratch
- append new channels
- read unsorted MDF v3 and v4 files
- filter a subset of channels from original mdf file
- cut measurement to specified time interval
- convert to different mdf version
- export to Excel, HDF5, Matlab and CSV
- merge multiple files sharing the same internal structure
- read and save mdf version 4.10 files containing zipped data blocks
- split large data blocks (configurable size) for mdf version 4
- disk space savings by compacting 1-dimensional integer channels (configurable)
- full support (read, append, save) for the following map types (multidimensional array channels):
 - mdf version 3 channels with CDBLOCK
 - mdf version 4 structure channel composition
 - mdf version 4 channel arrays with CNTemplate storage and one of the array types:
 - * 0 - array
 - * 1 - scaling axis
 - * 2 - look-up
- add and extract attachments for mdf version 4
- files are loaded in RAM for fast operations
- handle large files (exceeding the available RAM) using *memory = minimum* argument

- extract channel data, master channel and extra channel information as *Signal* objects for unified operations with v3 and v4 files
- time domain operation using the *Signal* class
 - Pandas data frames are good if all the channels have the same time based
 - usually a measurement will have channels from different sources at different rates
 - the *Signal* class facilitates operations with such channels

Major features not implemented (yet)

- for version 3
 - functionality related to sample reduction block (but the class is defined)
- for version 4
 - handling of bus logging measurements
 - handling of unfinished measurements (mdf 4)
 - full support for remaining mdf 4 channel arrays types
 - xml schema for TXBLOCK and MDBLOCK
 - partial conversions
 - event blocks
 - channels with default X axis
 - channels with reference to attachment

asammdf uses the following libraries

- numpy : the heart that makes all tick
- numexpr : for algebraic and rational channel conversions
- matplotlib : for Signal plotting
- wheel : for installation in virtual environments
- pandas : for DataFrame export

optional dependencies needed for exports

- h5py : for HDF5 export
- xlswriter : for Excel export
- scipy : for Matlab .mat export

CHAPTER 5

Installation

asammdf is available on

- github: <https://github.com/danielhrisca/asammdf/>
- PyPI: <https://pypi.org/project/asammdf/>

```
pip install asammdf
```


6.1 Package level

`asammdf.configure` (*integer_compacting=None, split_data_blocks=None, split_threshold=None, overwrite=None*)
configure asammdf parameters

Parameters `integer_compacting` : bool

enable/disable compacting of integer channels on append. This has the potential to greatly reduce file size, but append speed is slower and further loading of the resulting file will also be slower.

split_data_blocks : bool

enable/disable splitting of large data blocks using data lists for mdf version 4

split_threshold : int

size hint of splitted data blocks, default 2MB; if the initial size is smaller then no data list is used. The actual split size depends on the data groups' records size

overwrite : bool

default option for save method's overwrite argument

Enabling compacting of integer channels on append the file size of the resulting file can decrease up to a factor of ~0.5. Splitting the data blocks is usefull for large blocks. The recommended maximum threshold by ASAM is 4MB. *asammdf* uses a default of 2MB

6.2 MDF

This class acts as a proxy for the MDF3 and MDF4 classes. All attribute access is delegated to the underlying *_mdf* attribute (MDF3 or MDF4 object). See MDF3 and MDF4 for available extra methods.

An empty MDF file is created if the *name* argument is not provided. If the *name* argument is provided then the file must exist in the filesystem, otherwise an exception is raised.

Best practice is to use the MDF as a context manager. This way all resources are released correctly in case of exceptions.

```
with MDF(r'test.mdf') as mdf_file:
    # do something
```

class asammdf.mdf.MDF (*name=None, memory='full', version='4.10'*)

Unified access to MDF v3 and v4 files.

Parameters *name* : string

mdf file name, if provided it must be a real file name

memory : str

memory option; default *full*

- if *full* the data group binary data block will be loaded in RAM
- **if *low* the channel data is read from disk on request, and the** metadata is loaded into RAM
- if *minimum* only minimal data is loaded into RAM

version : string

mdf file version from ('2.00', '2.14', '3.00', '3.10', '3.20', '3.30', '4.00', '4.10', '4.11'); default '4.10'

convert (*to, memory='full'*)

convert MDF to other versions

Parameters *to* : str

new mdf version from ('2.00', '2.14', '3.00', '3.10', '3.20', '3.30', '4.00', '4.10', '4.11')

memory : str

memory option; default *full*

Returns *out* : MDF

new MDF object

cut (*start=None, stop=None, whence=0*)

convert MDF to other versions

Parameters *start* : float

start time, default *None*. If *None* then the start of measurement is used

stop : float

stop time, default . If *None* then the end of measurement is used

whence : int

how to search for the start and stop values

- 0 : absolute
- 1 : relative to first timestamp

Returns *out* : MDF

new MDF object

export (*fmt, filename=None*)

export MDF to other formats. The *MDF* file name is used is available, else the *filename* argument must be provided.

Parameters *fmt* : string

can be one of the following:

- *csv* [CSV export that uses the “;” delimiter. This option] will generate a new csv file for each data group (<MDFNAME>_DataGroup_<cntr>.csv)
- *hdf5* [HDF5 file output; each *MDF* data group is mapped to] a *HDF5* group with the name ‘DataGroup_<cntr>’ (where <cntr> is the index)
- *excel* [Excel file output (very slow). This option will] generate a new excel file for each data group (<MDFNAME>_DataGroup_<cntr>.xlsx)
- *mat* [Matlab .mat version 5 export, for Matlab >= 7.6. In] the mat file the channels will be renamed to ‘DataGroup_<cntr>_<channel name>’. The channel group master will be renamed to ‘DataGroup_<cntr>_<channel name>_master’ (<cntr> is the data group index starting from 0)

filename : string

export file name

filter (*channels, memory=None*)

return new *MDF* object that contains only the channels listed in *channels* argument

Parameters *channels* : list

list of channel names to be filtered

memory : str

memory option for filtered mdf; default None in which case the original file’s memory option is used

Returns *mdf* : MDF

new MDF file

iter_to_pandas ()

generator that yields channel groups as pandas DataFrames

static merge (*files, outversion='4.10', memory='full'*)

merge several files and return the merged MDF object. The files must have the same internal structure (same number of groups, and same channels in each group)

Parameters *files* : list | tuple

list of MDF file names

outversion : str

merged file version

memory : str

memory option; default *full*

Returns *merged* : MDF

new MDF object with merged channels

Raises **MdfException** : if there are inconsistencies between the files

merged MDF object

resample (*raster, memory=None*)

resample all channels to given raster

Parameters **raster** : float

time raster is seconds

memory : str

memory option; default *None*

Returns **mdf** : MDF

new MDF with resampled channels

select (*channels, dataframe=False*)

return the channels listed in *channels* argument

Parameters **channels** : list

list of channel names to be filtered

dataframe: bool

return a pandas DataFrame instead of a list of Signals; in this case the signals will be interpolated using the union of all timestamps

Returns **signals** : list

list of *Signal* objects based on the input channel list

6.2.1 MDF2, MDF3 and MDF4 classes

MDF2

asammdf tries to emulate the mdf structure using Python builtin data types.

The *header* attribute is an OrderedDict that holds the file metadata.

The *groups* attribute is a dictionary list with the following keys:

- **data_group** : DataGroup object
- **channel_group** : ChannelGroup object
- **channels** : list of Channel objects with the same order as found in the mdf file
- **channel_conversions** : list of ChannelConversion objects in 1-to-1 relation with the channel list
- **channel_sources** : list of SourceInformation objects in 1-to-1 relation with the channels list
- **channel_dependencies** : list of ChannelDependency objects in a 1-to-1 relation with the channel list
- **data_block** : DataBlock object
- **texts** : dictionary containing TextBlock objects used throughout the mdf
 - **channels** : list of dictionaries that contain TextBlock objects related to each channel
 - * **long_name_addr** : channel long name
 - * **comment_addr** : channel comment

- * `display_name_addr` : channel display name
- `channel_group` : list of dictionaries that contain TextBlock objects related to each channel group
 - * `comment_addr` : channel group comment
- `conversion_tab` : list of dictionaries that contain TextBlock objects related to VATB and VTABR channel conversions
 - * `text_{n}` : n-th text of the VTABR conversion
- `sorted` : bool flag to indicate if the source file was sorted; it is used when *memory* is *low* or *minimum*
- `size` : data block size; used for lazy loading of measured data
- `record_size` : dict of record ID -> record size pairs

The `file_history` attribute is a TextBlock object.

The `channel_db` attribute is a dictionary that holds the (*data group index*, *channel index*) pair for all signals. This is used to speed up the `get_signal_by_name` method.

The `master_db` attribute is a dictionary that holds the *channel index* of the master channel for all data groups. This is used to speed up the `get_signal_by_name` method.

API

class `asammdf.mdf2.MDF2` (*name=None*, *memory=2*, *version='2.14'*)

If the *name* exist it will be loaded otherwise an empty file will be created that can be later saved to disk

Parameters `name` : string

mdf file name

memory : str

memory optimization option; default *full*

- if *full* the data group binary data block will be memorised in RAM
- **if *low* the channel data is read from disk on request, and the** `metadata` **is** memorised into RAM
- if *minimum* only minimal data is memorised into RAM

version : string

mdf file version ('2.00' or '2.14'); default '2.14'

Attributes

name	(string) mdf file name
groups	(list) list of data groups
header	(OrderedDict) mdf file header
file_history	(TextBlock) file history text block; can be None
memory	(bool) load measured data option
version	(str) mdf version
channels_db	(dict) used for fast channel access by name; for each name key the value is a list of (group index, channel index) tuples
masters_db	(dict) used for fast master channel access; for each group index key the value is the master channel index

add_trigger (*group, timestamp, pre_time=0, post_time=0, comment=""*)
add trigger to data group

Parameters **group** : int
group index
timestamp : float
trigger time
pre_time : float
trigger pre time; default 0
post_time : float
trigger post time; default 0
comment : str
trigger comment

append (*signals, acquisition_info='Python', common_timebase=False*)
Appends a new data group.

For channel dependencies type Signals, the *samples* attribute must be a `numpy.recarray`

Parameters **signals** : list
list on *Signal* objects
acquisition_info : str
acquisition information; default 'Python'
common_timebase : bool
flag to hint that the signals have the same timebase

Examples

```
>>> # case 1 conversion type None
>>> s1 = np.array([1, 2, 3, 4, 5])
>>> s2 = np.array([-1, -2, -3, -4, -5])
>>> s3 = np.array([0.1, 0.04, 0.09, 0.16, 0.25])
>>> t = np.array([0.001, 0.002, 0.003, 0.004, 0.005])
>>> names = ['Positive', 'Negative', 'Float']
>>> units = ['+', '-', '.f']
>>> info = {}
>>> s1 = Signal(samples=s1, timestamps=t, unit='+', name='Positive')
>>> s2 = Signal(samples=s2, timestamps=t, unit='-', name='Negative')
>>> s3 = Signal(samples=s3, timestamps=t, unit='flts', name='Floats')
>>> mdf = MDF2('new.mdf')
>>> mdf.append([s1, s2, s3], 'created by asammdf v1.1.0')
>>> # case 2: VTAB conversions from channels inside another file
>>> mdf1 = MDF2('in.mdf')
>>> ch1 = mdf1.get("Channel1_VTAB")
>>> ch2 = mdf1.get("Channel2_VTABR")
>>> sigs = [ch1, ch2]
>>> mdf2 = MDF2('out.mdf')
>>> mdf2.append(sigs, 'created by asammdf v1.1.0')
```

close()

if the MDF was created with `memory='minimum'` and new channels have been appended, then this must be called just before the object is not used anymore to clean-up the temporary file

get (*name=None, group=None, index=None, raster=None, samples_only=False, data=None*)

Gets channel samples.

Channel can be specified in two ways:

- using the first positional argument *name*
 - **if there are multiple occurrences for this channel then the** *group* and *index* arguments can be used to select a specific group.
 - **if there are multiple occurrences for this channel and either the** *group* or *index* arguments is *None* then a warning is issued
- **using the group number (keyword argument *group*) and the channel number** (keyword argument *index*). Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters *name* : string

name of channel

group : int

0-based group index

index : int

0-based channel index

raster : float

time raster in seconds

samples_only : bool

if *True* return only the channel samples as numpy array; if *False* return a *Signal* object

Returns *res* : (numpy.array | *Signal*)

returns *Signal* if *samples_only* != *False* (default option), otherwise returns numpy.array. The *Signal* samples are:

- **numpy recarray for channels that have CDBLOCK or BYTEARRAY** type channels
- numpy array for all the rest

Raises *MdfError* :

- * **if the channel name is not found**
- * **if the group index is out of range**
- * **if the channel index is out of range**

get_channel_comment (*name=None, group=None, index=None*)

Gets channel comment.

Channel can be specified in two ways:

- using the first positional argument *name*

- if there are multiple occurrences for this channel then the *group* and *index* arguments can be used to select a specific group.
- if there are multiple occurrences for this channel and either the *group* or *index* arguments is *None* then a warning is issued
- using the group number (keyword argument *group*) and the channel number (keyword argument *index*). Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters *name* : string

name of channel

group : int

0-based group index

index : int

0-based channel index

Returns *comment* : str

found channel comment

get_channel_unit (*name=None, group=None, index=None*)

Gets channel unit.

Channel can be specified in two ways:

- using the first positional argument *name*
 - if there are multiple occurrences for this channel then the *group* and *index* arguments can be used to select a specific group.
 - if there are multiple occurrences for this channel and either the *group* or *index* arguments is *None* then a warning is issued
- using the group number (keyword argument *group*) and the channel number (keyword argument *index*). Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters *name* : string

name of channel

group : int

0-based group index

index : int

0-based channel index

Returns *unit* : str

found channel unit

get_master (*index, data=None*)

returns master channel samples for given group

Parameters *index* : int

group index

data : bytes

data block raw bytes; default None

Returns **t** : numpy.array

master channel samples

info()

get MDF information as a dict

Examples

```
>>> mdf = MDF2('test.mdf')
>>> mdf.info()
```

iter_get_triggers()

generator that yields triggers

Returns **trigger_info** : dict

trigger information with the following keys:

- comment : trigger comment
- time : trigger time
- pre_time : trigger pre time
- post_time : trigger post time
- index : trigger index
- group : data group index of trigger

save (*dst=""*, *overwrite=None*, *compression=0*)

Save MDF to *dst*. If *dst* is not provided the the destination file name is the MDF name. If *overwrite* is *True* then the destination file is overwritten, otherwise the file name is appended with '*<cntr>*', were '*<cntr>*' is the first counter that produces a new file name (that does not already exist in the filesystem).

Parameters **dst** : str

destination file name, Default ''

overwrite : bool

overwrite flag, default *False*

compression : int

does nothing for mdf version3; introduced here to share the same API as mdf version 4 files

MDF version 2 blocks

The following classes implement different MDF version3 blocks.

Channel Class

class `asammdf.mdf2.Channel` (***kargs*)

CNBLOCK class derived from *dict*

The Channel object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using any of the following presented keys - when creating a new Channel

The keys have the following meaning:

- **id** - Block type identifier, always “CN”
- **block_len** - Block size of this block in bytes (entire CNBLOCK)
- **next_ch_addr** - Pointer to next channel block (CNBLOCK) of this channel group (NIL allowed)
- **conversion_addr** - Pointer to the conversion formula (CCBLOCK) of this signal (NIL allowed)
- **source_depend_addr** - Pointer to the source-depending extensions (CEBLOCK) of this signal (NIL allowed)
- **ch_depend_addr** - Pointer to the dependency block (CDBLOCK) of this signal (NIL allowed)
- **comment_addr** - Pointer to the channel comment (TXBLOCK) of this signal (NIL allowed)
- **channel_type** - Channel type
 - 0 = data channel
 - 1 = time channel for all signals of this group (in each channel group, exactly one channel must be defined as time channel). The time stamps recording in a time channel are always relative to the start time of the measurement defined in HDBLOCK.
- **short_name** - Short signal name, i.e. the first 31 characters of the ASAM-MCD name of the signal (end of text should be indicated by 0)
- **description** - Signal description (end of text should be indicated by 0)
- **start_offset** - Start offset in bits to determine the first bit of the signal in the data record. The start offset N is divided into two parts: a “Byte offset” ($= N \div 8$) and a “Bit offset” ($= N \bmod 8$). The channel block can define an “additional Byte offset” (see below) which must be added to the Byte offset.
- **bit_count** - Number of bits used to encode the value of this signal in a data record
- **data_type** - Signal data type
- **range_flag** - Value range valid flag
- **min_raw_value** - Minimum signal value that occurred for this signal (raw value)
- **max_raw_value** - Maximum signal value that occurred for this signal (raw value)
- **sampling_rate** - Sampling rate for a virtual time channel. Unit [s]
- **long_name_addr** - Pointer to TXBLOCK that contains the ASAM-MCD long signal name
- **display_name_addr** - Pointer to TXBLOCK that contains the signal’s display name (NIL allowed)
- **additional_byte_offset** - Additional Byte offset of the signal in the data record (default value: 0).

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     ch1 = Channel(stream=mdf, address=0xBA52)
>>> ch2 = Channel()
>>> ch1.name
'VehicleSpeed'
>>> ch1['id']
b'CN'
```

Attributes

name	(str) full channel name
address	(int) block address inside mdf file
dependencies	(list) list of channel dependencies

ChannelConversion Class

class asammdf.mdf2.ChannelConversion (**kargs)

CCBLOCK class derived from *dict*

The ChannelConversion object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using any of the following presented keys - when creating a new ChannelConversion

The first keys are common for all conversion types, and are followed by conversion specific keys. The keys have the following meaning:

- common keys
 - id - Block type identifier, always “CC”
 - block_len - Block size of this block in bytes (entire CCBLOCK)
 - range_flag - Physical value range valid flag:
 - min_phy_value - Minimum physical signal value that occurred for this signal
 - max_phy_value - Maximum physical signal value that occurred for this signal
 - unit - Physical unit (string should be terminated with 0)
 - conversion_type - Conversion type (formula identifier)
 - ref_param_nr - Size information about additional conversion data
- specific keys
 - linear conversion
 - * b - offset
 - * a - factor
 - * **CANapeHiddenExtra** - sometimes CANape appends extra information; not compliant with MDF specs
 - ASAM formula conversion

- * formula - equation as string
- polynomial or rational conversion
 - * P1 .. P6 - factors
- exponential or logarithmic conversion
 - * P1 .. P7 - factors
- tabular with or without interpolation (grouped by *n*)
 - * raw_{n} - n-th raw integer value (X axis)
 - * phys_{n} - n-th physical value (Y axis)
- text table conversion
 - * param_val_{n} - n-th integers value (X axis)
 - * text_{n} - n-th text value (Y axis)
- text range table conversion
 - * lower_{n} - n-th lower raw value
 - * upper_{n} - n-th upper raw value
 - * text_{n} - n-th text value

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     cc1 = ChannelConversion(stream=mdf, address=0xBA52)
>>> cc2 = ChannelConversion(conversion_type=0)
>>> cc1['b'], cc1['a']
0, 100.0
```

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

ChannelDependency Class

class `asammdf.mdf2.ChannelDependency` (**kargs)
 CDBLOCK class derived from *dict*

Currently the ChannelDependency object can only be created using the *stream* and *address* keyword parameters when reading from file

The keys have the following meaning:

- `id` - Block type identifier, always “CD”
- `block_len` - Block size of this block in bytes (entire CDBLOCK)
- `dependency_type` - Dependency type
- `sd_nr` - Total number of signals dependencies (m)
- for each dependency there is a group of three keys:
 - `dg_{n}` - Pointer to the data group block (DGBLOCK) of signal dependency n
 - `cg_{n}` - Pointer to the channel group block (DGBLOCK) of signal dependency n
 - `ch_{n}` - Pointer to the channel block (DGBLOCK) of signal dependency n
- there can also be optional keys which describe dimensions for the N-dimensional dependencies:
 - `dim_{n}` - Optional: size of dimension n for N-dimensional dependency

Parameters `stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

ChannelExtension Class

class `asammdf.mdf2.ChannelExtension` (**kargs)

CEBLOCK class derived from *dict*

The ChannelExtension object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using any of the following presented keys - when creating a new ChannelExtension

The first keys are common for all conversion types, and are followed by conversion specific keys. The keys have the following meaning:

- common keys
 - `id` - Block type identifier, always “CE”
 - `block_len` - Block size of this block in bytes (entire CEBLOCK)
 - `type` - Extension type identifier
- specific keys
 - for DIM block
 - * `module_nr` - Number of module
 - * `module_address` - Address
 - * `description` - Description

- * ECU_identification - Identification of ECU
- * reserved0' - reserved
- for Vector CAN block
 - * CAN_id - Identifier of CAN message
 - * CAN_ch_index - Index of CAN channel
 - * message_name - Name of message (string should be terminated by 0)
 - * sender_name - Name of sender (string should be terminated by 0)
 - * reserved0 - reserved

Parameters `stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

ChannelGroup Class

class `asammdf.mdf2.ChannelGroup` (***kargs*)
 CGBLOCK class derived from *dict*

The ChannelGroup object can be created in two modes:

- **using the *stream* and *address* keyword parameters - when reading from file**
- **using any of the following presented keys - when creating a new ChannelGroup**

The keys have the following meaning:

- **id** - Block type identifier, always “CG”
- **block_len** - Block size of this block in bytes (entire CGBLOCK)
- **next_cg_addr** - Pointer to next channel group block (CGBLOCK) (NIL allowed)
- **first_ch_addr** - Pointer to first channel block (CNBLOCK) (NIL allowed)
- **comment_addr** - **Pointer to channel group comment text (TXBLOCK)** (NIL allowed)
- **record_id** - **Record ID, i.e. value of the identifier for a record if the DGBLOCK defines a number of record IDs > 0**
- **ch_nr** - Number of channels (redundant information)
- **samples_byte_nr** - **Size of data record in Bytes (without record ID), i.e.** size of plain data for a each recorded sample of this channel group
- **cycles_nr** - **Number of records of this type in the data block i.e. number** of samples for this channel group

- **sample_reduction_addr** - only since version 3.3. Pointer to first sample reduction block (SR-BLOCK) (NIL allowed) Default value: NIL.

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     cg1 = ChannelGroup(stream=mdf, address=0xBA52)
>>> cg2 = ChannelGroup(sample_bytes_nr=32)
>>> hex(cg1.address)
0xBA52
>>> cg1['id']
b'CG'
```

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

DataGroup Class

class asammdf.mdf2.**DataGroup** (**kargs)

DGBLOCK class derived from *dict*

The DataGroup object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using any of the following presented keys - when creating a new DataGroup

The keys have the following meaning:

- **id** - Block type identifier, always “DG”
- **block_len** - Block size of this block in bytes (entire DGBLOCK)
- **next_dg_addr** - Pointer to next data group block (DGBLOCK) (NIL allowed)
- **first_cg_addr** - Pointer to first channel group block (CGBLOCK) (NIL allowed)
- **trigger_addr** - Pointer to trigger block (TRBLOCK) (NIL allowed)
- **data_block_addr** - Pointer to the data block
- **cg_nr** - Number of channel groups (redundant information)
- **record_id_nr** - Number of record IDs in the data block
- **reserved0** - since version 3.2; Reserved

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

FileIdentificationBlock Class

class `asammdf.mdf2.FileIdentificationBlock(**kargs)`

IDBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using the classmethod *from_text*

The keys have the following meaning:

- *file_identification* - file identifier
- *version_str* - format identifier
- *program_identification* - program identifier
- *byte_order* - default byte order
- *float_format* - default floating-point format
- *mdf_version* - version number of MDF format
- *code_page* - code page number
- *reserved0* - reserved
- *reserved1* - reserved
- *unfinalized_standard_flags* - Standard Flags for unfinalized MDF
- *unfinalized_custom_flags* - Custom Flags for unfinalized MDF

Parameters *stream* : file handle

mdf file handle

version : int

mdf version in case of new file

Attributes

address	(int) block address inside mdf file; should be 0 always
----------------	---

HeaderBlock Class

class asammdf.mdf2.**HeaderBlock** (**kargs)

HDBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the *stream* - when reading from file
- using the classmethod *from_text*

The keys have the following meaning:

- **id** - Block type identifier, always “HD”
- **block_len** - Block size of this block in bytes (entire HDBLOCK)
- **first_dg_addr** - Pointer to the first data group block (DGBLOCK)
- **comment_addr** - **Pointer to the measurement file comment text (TXBLOCK) (NIL allowed)**
- **program_addr** - Pointer to program block (PRBLOCK) (NIL allowed)
- **dg_nr** - Number of data groups (redundant information)
- **date** - Date at which the recording was started in “DD:MM:YYYY” format
- **time** - Time at which the recording was started in “HH:MM:SS” format
- **author** - author name
- **organization** - organization
- **project** - project name
- **subject** - subject

Since version 3.2 the following extra keys were added:

- **abs_time** - Time stamp at which recording was started in nanoseconds.
- **tz_offset** - UTC time offset in hours (= GMT time zone)
- **time_quality** - Time quality class
- **timer_identification** - Timer identification (time source),

Parameters **stream** : file handle

mdf file handle

Attributes

address	(int) block address inside mdf file; should be 64 always
----------------	--

ProgramBlock Class

class asammdf.v2blocks.**ProgramBlock** (**kargs)

PRBLOCK class derived from *dict*

The ProgramBlock object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file

- using any of the following presented keys - when creating a new ProgramBlock

The keys have the following meaning:

- `id` - Block type identifier, always “PR”
- `block_len` - Block size of this block in bytes (entire PRBLOCK)
- `data` - Program-specific data

Parameters `stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

SampleReduction Class

class `asammdf.v2blocks.SampleReduction` (**kargs)

SRBLOCK class derived from *dict*

Currently the SampleReduction object can only be created by using the *stream* and *address* keyword parameters
- when reading from file

The keys have the following meaning:

- `id` - Block type identifier, always “SR”
- `block_len` - Block size of this block in bytes (entire SRBLOCK)
- **`next_sr_addr` - Pointer to next sample reduction block (SRBLOCK) (NIL allowed)**
- `data_block_addr` - Pointer to the data block for this sample reduction
- `cycles_nr` - Number of reduced samples in the data block.
- **`time_interval` - Length of time interval [s] used to calculate the reduced samples.**

Parameters `stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

TextBlock Class

class asammdf.mdf2.**TextBlock** (**kargs)
 TXBLOCK class derived from *dict*

The ProgramBlock object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using the classmethod *from_text*

The keys have the following meaning:

- *id* - Block type identifier, always “TX”
- *block_len* - Block size of this block in bytes (entire TXBLOCK)
- *text* - Text (new line indicated by CR and LF; end of text indicated by 0)

Parameters *stream* : file handle

mdf file handle

address : int

block address inside mdf file

text : bytes

bytes for creating a new TextBlock

Examples

```
>>> tx1 = TextBlock.from_text('VehicleSpeed')
>>> tx1.text_str
'VehicleSpeed'
>>> tx1['text']
b'VehicleSpeed'
```

Attributes

address	(int) block address inside mdf file
text_str	(str) text data as unicode string

TriggerBlock Class

class asammdf.mdf2.**TriggerBlock** (**kargs)
 TRBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using the classmethod *from_text*

The keys have the following meaning:

- *id* - Block type identifier, always “TX”

- `block_len` - Block size of this block in bytes (entire TRBLOCK)
- `text_addr` - Pointer to trigger comment text (TXBLOCK) (NIL allowed)
- `trigger_events_nr` - Number of trigger events n (0 allowed)
- `trigger_{n}_time` - Trigger time [s] of trigger event n
- `trigger_{n}_pretime` - Pre trigger time [s] of trigger event n
- `trigger_{n}_posttime` - Post trigger time [s] of trigger event n

Parameters `stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

MDF3

asammdf tries to emulate the mdf structure using Python builtin data types.

The *header* attribute is an OrderedDict that holds the file metadata.

The *groups* attribute is a dictionary list with the following keys:

- `data_group` : DataGroup object
- `channel_group` : ChannelGroup object
- `channels` : list of Channel objects with the same order as found in the mdf file
- `channel_conversions` : list of ChannelConversion objects in 1-to-1 relation with the channel list
- `channel_sources` : list of SourceInformation objects in 1-to-1 relation with the channels list
- `channel_dependencies` : list of ChannelDependency objects in a 1-to-1 relation with the channel list
- `data_block` : DataBlock object
- `texts` : dictionary containing TextBlock objects used throughout the mdf
 - `channels` : list of dictionaries that contain TextBlock objects related to each channel
 - * `long_name_addr` : channel long name
 - * `comment_addr` : channel comment
 - * `display_name_addr` : channel display name
 - `channel group` : list of dictionaries that contain TextBlock objects related to each channel group
 - * `comment_addr` : channel group comment
 - `conversion_tab` : list of dictionaries that contain TextBlock objects related to VATB and VTABR channel conversions
 - * `text_{n}` : n -th text of the VTABR conversion

- `sorted` : bool flag to indicate if the source file was sorted; it is used when *memory* is *low* or *minimum*
- `size` : data block size; used for lazy loading of measured data
- `record_size` : dict of record ID -> record size pairs

The `file_history` attribute is a `TextBlock` object.

The `channel_db` attribute is a dictionary that holds the (*data group index*, *channel index*) pair for all signals. This is used to speed up the `get_signal_by_name` method.

The `master_db` attribute is a dictionary that holds the *channel index* of the master channel for all data groups. This is used to speed up the `get_signal_by_name` method.

API

class `asammdf.mdf3.MDF3` (*name=None*, *memory=2*, *version='3.30'*)

If the *name* exist it will be loaded otherwise an empty file will be created that can be later saved to disk

Parameters `name` : string

mdf file name

memory : str

memory optimization option; default *full*

- if *full* the data group binary data block will be memorised in RAM
- if **low** the channel data is read from disk on request, and the metadata is memorised into RAM
- if *minimum* only minimal data is memorised into RAM

version : string

mdf file version ('3.00', '3.10', '3.20' or '3.30'); default '3.30'

Attributes

name	(string) mdf file name
groups	(list) list of data groups
header	(OrderedDict) mdf file header
file_history	(TextBlock) file history text block; can be None
memory	(bool) load measured data option
version	(str) mdf version
channels_db	(dict) used for fast channel access by name; for each name key the value is a list of (group index, channel index) tuples
masters_db	(dict) used for fast master channel access; for each group index key the value is the master channel index

add_trigger (*group*, *timestamp*, *pre_time=0*, *post_time=0*, *comment=""*)

add trigger to data group

Parameters `group` : int

group index

timestamp : float

trigger time
pre_time : float
trigger pre time; default 0
post_time : float
trigger post time; default 0
comment : str
trigger comment

append (*signals*, *acquisition_info*='Python', *common_timebase*=False)

Appends a new data group.

For channel dependencies type Signals, the *samples* attribute must be a `numpy.recarray`

Parameters **signals** : list
list on *Signal* objects
acquisition_info : str
acquisition information; default 'Python'
common_timebase : bool
flag to hint that the signals have the same timebase

Examples

```
>>> # case 1 conversion type None
>>> s1 = np.array([1, 2, 3, 4, 5])
>>> s2 = np.array([-1, -2, -3, -4, -5])
>>> s3 = np.array([0.1, 0.04, 0.09, 0.16, 0.25])
>>> t = np.array([0.001, 0.002, 0.003, 0.004, 0.005])
>>> names = ['Positive', 'Negative', 'Float']
>>> units = ['+', '-', '.f']
>>> info = {}
>>> s1 = Signal(samples=s1, timestamps=t, unit='+', name='Positive')
>>> s2 = Signal(samples=s2, timestamps=t, unit='-', name='Negative')
>>> s3 = Signal(samples=s3, timestamps=t, unit='flts', name='Floats')
>>> mdf = MDF3('new.mdf')
>>> mdf.append([s1, s2, s3], 'created by asammdf v1.1.0')
>>> # case 2: VTAB conversions from channels inside another file
>>> mdf1 = MDF3('in.mdf')
>>> ch1 = mdf1.get("Channel1_VTAB")
>>> ch2 = mdf1.get("Channel2_VTABR")
>>> sigs = [ch1, ch2]
>>> mdf2 = MDF3('out.mdf')
>>> mdf2.append(sigs, 'created by asammdf v1.1.0')
```

close ()

if the MDF was created with `memory='minimum'` and new channels have been appended, then this must be called just before the object is not used anymore to clean-up the temporary file

get (*name*=None, *group*=None, *index*=None, *raster*=None, *samples_only*=False, *data*=None)

Gets channel samples. Channel can be specified in two ways:

- using the first positional argument *name*

- if there are multiple occurrences for this channel then the *group* and *index* arguments can be used to select a specific group.
- if there are multiple occurrences for this channel and either the *group* or *index* arguments is *None* then a warning is issued
- using the group number (keyword argument *group*) and the channel number (keyword argument *index*). Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters *name* : string

name of channel

group : int

0-based group index

index : int

0-based channel index

raster : float

time raster in seconds

samples_only : bool

if *True* return only the channel samples as numpy array; if *False* return a *Signal* object

Returns *res* : (numpy.array | *Signal*)

returns *Signal* if *samples_only* != *False* (default option), otherwise returns numpy.array. The *Signal* samples are:

- numpy recarray for channels that have CDBLOCK or BYTEARRAY type channels
- numpy array for all the rest

Raises *MdfError* :

- * if the channel name is not found
- * if the group index is out of range
- * if the channel index is out of range

get_channel_comment (*name=None, group=None, index=None*)

Gets channel comment. Channel can be specified in two ways:

- using the first positional argument *name*
 - if there are multiple occurrences for this channel then the *group* and *index* arguments can be used to select a specific group.
 - if there are multiple occurrences for this channel and either the *group* or *index* arguments is *None* then a warning is issued
- using the group number (keyword argument *group*) and the channel number (keyword argument *index*). Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters *name* : string

name of channel

group : int

0-based group index

index : int

0-based channel index

Returns comment : str

found channel comment

get_channel_unit (*name=None, group=None, index=None*)

Gets channel unit.

Channel can be specified in two ways:

- using the first positional argument *name*
 - **if there are multiple occurrences for this channel then the *group* and *index* arguments can be used to select a specific group.**
 - **if there are multiple occurrences for this channel and either the *group* or *index* arguments is None then a warning is issued**
- **using the group number (keyword argument *group*) and the channel number (keyword argument *index*).** Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters name : string

name of channel

group : int

0-based group index

index : int

0-based channel index

Returns unit : str

found channel unit

get_master (*index, data=None*)

returns master channel samples for given group

Parameters index : int

group index

data : bytes

data block raw bytes; default None

Returns t : numpy.array

master channel samples

info ()

get MDF information as a dict

Examples

```
>>> mdf = MDF3('test.mdf')
>>> mdf.info()
```

`iter_get_triggers()`

generator that yields triggers

Returns `trigger_info` : dict

trigger information with the following keys:

- `comment` : trigger comment
- `time` : trigger time
- `pre_time` : trigger pre time
- `post_time` : trigger post time
- `index` : trigger index
- `group` : data group index of trigger

save (*dst*=", *overwrite*=None, *compression*=0)

Save MDF to *dst*. If *dst* is not provided the the destination file name is the MDF name. If *overwrite* is *True* then the destination file is overwritten, otherwise the file name is appened with '`_<cntr>`', were '`<cntr>`' is the first conter that produces a new file name (that does not already exist in the filesystem).

Parameters `dst` : str

destination file name, Default ''

overwrite : bool

overwrite flag, default *False*

compression : int

does nothing for mdf version3; introduced here to share the same API as mdf version 4 files

MDF version 3 blocks

The following classes implement different MDF version3 blocks.

Channel Class

class `asammdf.mdf3.Channel` (***kargs*)

CNBLOCK class derived from *dict*

The Channel object can be created in two modes:

- **using the *stream* and *address* keyword parameters - when reading** from file
- using any of the following presented keys - when creating a new Channel

The keys have the following meaning:

- `id` - Block type identifier, always "CN"
- `block_len` - Block size of this block in bytes (entire CNBLOCK)

- **next_ch_addr** - Pointer to next channel block (CNBLOCK) of this channel group (NIL allowed)
- **conversion_addr** - Pointer to the conversion formula (CCBLOCK) of this signal (NIL allowed)
- **source_depend_addr** - Pointer to the source-depending extensions (CEBLOCK) of this signal (NIL allowed)
- **ch_depend_addr** - Pointer to the dependency block (CDBLOCK) of this signal (NIL allowed)
- **comment_addr** - Pointer to the channel comment (TXBLOCK) of this signal (NIL allowed)
- **channel_type** - Channel type
 - 0 = data channel
 - 1 = time channel for all signals of this group (in each channel group, exactly one channel must be defined as time channel). The time stamps recording in a time channel are always relative to the start time of the measurement defined in HDBLOCK.
- **short_name** - Short signal name, i.e. the first 31 characters of the ASAM-MCD name of the signal (end of text should be indicated by 0)
- **description** - Signal description (end of text should be indicated by 0)
- **start_offset** - Start offset in bits to determine the first bit of the signal in the data record. The start offset N is divided into two parts: a “Byte offset” ($= N \div 8$) and a “Bit offset” ($= N \bmod 8$). The channel block can define an “additional Byte offset” (see below) which must be added to the Byte offset.
- **bit_count** - Number of bits used to encode the value of this signal in a data record
- **data_type** - Signal data type
- **range_flag** - Value range valid flag
- **min_raw_value** - Minimum signal value that occurred for this signal (raw value)
- **max_raw_value** - Maximum signal value that occurred for this signal (raw value)
- **sampling_rate** - Sampling rate for a virtual time channel. Unit [s]
- **long_name_addr** - Pointer to TXBLOCK that contains the ASAM-MCD long signal name
- **display_name_addr** - Pointer to TXBLOCK that contains the signal’s display name (NIL allowed)
- **additional_byte_offset** - Additional Byte offset of the signal in the data record (default value: 0).

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     ch1 = Channel(stream=mdf, address=0xBA52)
>>> ch2 = Channel()
>>> ch1.name
'VehicleSpeed'
>>> ch1['id']
b'CN'
```

Attributes

name	(str) full channel name
address	(int) block address inside mdf file
dependencies	(list) list of channel dependencies

ChannelConversion Class

class `asammdf.mdf3.ChannelConversion` (***kargs*)

CCBLOCK class derived from *dict*

The ChannelConversion object can be created in two modes:

- **using the *stream* and *address* keyword parameters - when reading** from file
- **using any of the following presented keys - when creating a new** ChannelConversion

The first keys are common for all conversion types, and are followed by conversion specific keys. The keys have the following meaning:

- common keys
 - **id** - Block type identifier, always “CC”
 - **block_len** - Block size of this block in bytes (entire CCBLOCK)
 - **range_flag** - Physical value range valid flag:
 - **min_phy_value** - **Minimum physical signal value that occurred for this** signal
 - **max_phy_value** - **Maximum physical signal value that occurred for this** signal
 - **unit** - Physical unit (string should be terminated with 0)
 - **conversion_type** - Conversion type (formula identifier)
 - **ref_param_nr** - Size information about additional conversion data
- specific keys
 - linear conversion
 - * **b** - offset
 - * **a** - factor
 - * **CANapeHiddenExtra** - **sometimes CANape appends extra information;** not compliant with MDF specs
 - ASAM formula conversion
 - * **formula** - equation as string
 - polynomial or rational conversion
 - * **P1 .. P6** - factors
 - exponential or logarithmic conversion
 - * **P1 .. P7** - factors
 - tabular with or without interpolation (grouped by *n*)
 - * **raw_{n}** - *n*-th raw integer value (X axis)

- * `phys_{n}` - n-th physical value (Y axis)
- text table conversion
 - * `param_val_{n}` - n-th integers value (X axis)
 - * `text_{n}` - n-th text value (Y axis)
- text range table conversion
 - * `lower_{n}` - n-th lower raw value
 - * `upper_{n}` - n-th upper raw value
 - * `text_{n}` - n-th text value

Parameters `stream` : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     cc1 = ChannelConversion(stream=mdf, address=0xBA52)
>>> cc2 = ChannelConversion(conversion_type=0)
>>> cc1['b'], cc1['a']
0, 100.0
```

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

ChannelDependency Class

class `asammdf.mdf3.ChannelDependency` (***kargs*)

CDBLOCK class derived from *dict*

Currently the ChannelDependency object can only be created using the *stream* and *address* keyword parameters when reading from file

The keys have the following meaning:

- `id` - Block type identifier, always “CD”
- `block_len` - Block size of this block in bytes (entire CDBLOCK)
- `dependency_type` - Dependency type
- `sd_nr` - Total number of signals dependencies (m)
- for each dependency there is a group of three keys:
 - `dg_{n}` - Pointer to the data group block (DGBLOCK) of signal dependency *n*
 - `cg_{n}` - Pointer to the channel group block (DGBLOCK) of signal dependency *n*

- **ch_{n}** - Pointer to the channel block (DGBLOCK) of signal dependency *n*
- **there can also be optional keys which describe dimensions for** the N-dimensional dependencies:
 - **dim_{n}** - Optional: size of dimension *n* for N-dimensional dependency

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

ChannelExtension Class

class `asammdf.mdf3.ChannelExtension` (**kargs)

CEBLOCK class derived from *dict*

The ChannelExtension object can be created in two modes:

- **using the *stream* and *address* keyword parameters - when reading** from file
- **using any of the following presented keys - when creating** a new ChannelExtension

The first keys are common for all conversion types, and are followed by conversion specific keys. The keys have the following meaning:

- common keys
 - **id** - Block type identifier, always “CE”
 - **block_len** - Block size of this block in bytes (entire CEBLOCK)
 - **type** - Extension type identifier
- specific keys
 - for DIM block
 - * **module_nr** - Number of module
 - * **module_address** - Address
 - * **description** - Description
 - * **ECU_identification** - Identification of ECU
 - * **reserved0'** - reserved
 - for Vector CAN block
 - * **CAN_id** - Identifier of CAN message
 - * **CAN_ch_index** - Index of CAN channel
 - * **message_name** - Name of message (string should be terminated by 0)
 - * **sender_name** - Name of sender (string should be terminated by 0)

* reserved0 - reserved

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

ChannelGroup Class

class `asammdf.mdf3.ChannelGroup` (**kargs)

CGBLOCK class derived from *dict*

The ChannelGroup object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using any of the following presented keys - when creating a new ChannelGroup

The keys have the following meaning:

- **id** - Block type identifier, always “CG”
- **block_len** - Block size of this block in bytes (entire CGBLOCK)
- **next_cg_addr** - Pointer to next channel group block (CGBLOCK) (NIL allowed)
- **first_ch_addr** - Pointer to first channel block (CNBLOCK) (NIL allowed)
- **comment_addr** - Pointer to channel group comment text (TXBLOCK) (NIL allowed)
- **record_id** - Record ID, i.e. value of the identifier for a record if the DGBLOCK defines a number of record IDs > 0
- **ch_nr** - Number of channels (redundant information)
- **samples_byte_nr** - Size of data record in Bytes (without record ID), i.e. size of plain data for a each recorded sample of this channel group
- **cycles_nr** - Number of records of this type in the data block i.e. number of samples for this channel group
- **sample_reduction_addr** - only since version 3.3. Pointer to first sample reduction block (SRBLOCK) (NIL allowed) Default value: NIL.

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     cg1 = ChannelGroup(stream=mdf, address=0xBA52)
>>> cg2 = ChannelGroup(sample_bytes_nr=32)
>>> hex(cg1.address)
0xBA52
>>> cg1['id']
b'CG'
```

Attributes

address	(int) block address inside md5 file
----------------	-------------------------------------

DataGroup Class

class `asammdf.mdf3.DataGroup` (***kargs*)
 DGBLOCK class derived from *dict*

The DataGroup object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using any of the following presented keys - when creating a new DataGroup

The keys have the following meaning:

- **id** - Block type identifier, always “DG”
- **block_len** - Block size of this block in bytes (entire DGBLOCK)
- **next_dg_addr** - Pointer to next data group block (DGBLOCK) (NIL allowed)
- **first_cg_addr** - Pointer to first channel group block (CGBLOCK) (NIL allowed)
- **trigger_addr** - Pointer to trigger block (TRBLOCK) (NIL allowed)
- **data_block_addr** - Pointer to the data block (see separate chapter on data storage)
- **cg_nr** - Number of channel groups (redundant information)
- **record_id_nr** - Number of record IDs in the data block
- **reserved0** - since version 3.2; Reserved

Parameters **stream** : file handle

mdf file handle

address : int

block address inside md5 file

Attributes

address	(int) block address inside md5 file
----------------	-------------------------------------

FileIdentificationBlock Class

class `asammdf.mdf3.FileIdentificationBlock` (**kargs)
IDBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the *stream* and *address* keyword parameters - when reading from file
- using the classmethod *from_text*

The keys have the following meaning:

- `file_identification` - file identifier
- `version_str` - format identifier
- `program_identification` - program identifier
- `byte_order` - default byte order
- `float_format` - default floating-point format
- `mdf_version` - version number of MDF format
- `code_page` - code page number
- `reserved0` - reserved
- `reserved1` - reserved
- `unfinalized_standard_flags` - Standard Flags for unfinalized MDF
- `unfinalized_custom_flags` - Custom Flags for unfinalized MDF

Parameters `stream` : file handle

mdf file handle

version : int

mdf version in case of new file

Attributes

address	(int) block address inside mdf file; should be 0 always
----------------	---

HeaderBlock Class

class `asammdf.mdf3.HeaderBlock` (**kargs)
HDBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the *stream* - when reading from file
- using the classmethod *from_text*

The keys have the following meaning:

- `id` - Block type identifier, always “HD”
- `block_len` - Block size of this block in bytes (entire HDBLOCK)

- `first_dg_addr` - Pointer to the first data group block (DGBLOCK)
- **`comment_addr` - Pointer to the measurement file comment text (TXBLOCK)** (NIL allowed)
- `program_addr` - Pointer to program block (PRBLOCK) (NIL allowed)
- `dg_nr` - Number of data groups (redundant information)
- `date` - Date at which the recording was started in “DD:MM:YYYY” format
- `time` - Time at which the recording was started in “HH:MM:SS” format
- `author` - author name
- `organization` - organization
- `project` - project name
- `subject` - subject

Since version 3.2 the following extra keys were added:

- `abs_time` - Time stamp at which recording was started in nanoseconds.
- `tz_offset` - UTC time offset in hours (= GMT time zone)
- `time_quality` - Time quality class
- `timer_identification` - Timer identification (time source),

Parameters `stream` : file handle

mdf file handle

Attributes

address	(int) block address inside mdf file; should be 64 always
----------------	--

ProgramBlock Class

class `asammdf.v3blocks.ProgramBlock` (**kargs)

PRBLOCK class derived from *dict*

The ProgramBlock object can be created in two modes:

- **using the *stream* and *address* keyword parameters - when reading** from file
- **using any of the following presented keys - when creating** a new ProgramBlock

The keys have the following meaning:

- `id` - Block type identifier, always “PR”
- `block_len` - Block size of this block in bytes (entire PRBLOCK)
- `data` - Program-specific data

Parameters `stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

SampleReduction Class

class `asammdf.v3blocks.SampleReduction` (**kargs)

SRBLOCK class derived from *dict*

Currently the SampleReduction object can only be created by using the *stream* and *address* keyword parameters - when reading from file

The keys have the following meaning:

- **id** - Block type identifier, always “SR”
- **block_len** - Block size of this block in bytes (entire SRBLOCK)
- **next_sr_addr** - **Pointer to next sample reduction block (SRBLOCK)** (NIL allowed)
- **data_block_addr** - Pointer to the data block for this sample reduction
- **cycles_nr** - Number of reduced samples in the data block.
- **time_interval** - **Length of time interval [s] used to calculate** the reduced samples.

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

TextBlock Class

class `asammdf.mdf3.TextBlock` (**kargs)

TXBLOCK class derived from *dict*

The ProgramBlock object can be created in two modes:

- **using the *stream* and *address* keyword parameters - when reading** from file
- using the classmethod *from_text*

The keys have the following meaning:

- **id** - Block type identifier, always “TX”
- **block_len** - Block size of this block in bytes (entire TXBLOCK)

- **text** - Text (new line indicated by CR and LF; end of text indicated by 0)

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

text : bytes

bytes for creating a new TextBlock

Examples

```
>>> tx1 = TextBlock.from_text('VehicleSpeed')
>>> tx1.text_str
'VehicleSpeed'
>>> tx1['text']
b'VehicleSpeed'
```

Attributes

address	(int) block address inside mdf file
text_str	(str) text data as unicode string

TriggerBlock Class

class asammdf.mdf3.**TriggerBlock** (**kargs)

TRBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the **stream** and **address** keyword parameters - when reading from file
- using the classmethod *from_text*

The keys have the following meaning:

- **id** - Block type identifier, always “TX”
- **block_len** - Block size of this block in bytes (entire TRBLOCK)
- **text_addr** - Pointer to trigger comment text (TXBLOCK) (NIL allowed)
- **trigger_events_nr** - Number of trigger events *n* (0 allowed)
- **trigger_{n}_time** - Trigger time [s] of trigger event *n*
- **trigger_{n}_pretime** - Pre trigger time [s] of trigger event *n*
- **trigger_{n}_posttime** - Post trigger time [s] of trigger event *n*

Parameters **stream** : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

address	(int) block address inside mdf file
----------------	-------------------------------------

MDF4

asammdf tries to emulate the mdf structure using Python builtin data types.

The *header* attribute is an OrderedDict that holds the file metadata.

The *groups* attribute is a dictionary list with the following keys:

- **data_group** : DataGroup object
- **channel_group** : ChannelGroup object
- **channels** : list of Channel objects with the same order as found in the mdf file
- **channel_conversions** : list of ChannelConversion objects in 1-to-1 relation with the channel list
- **channel_sources** : list of SourceInformation objects in 1-to-1 relation with the channels list
- **data_block** : DataBlock object
- **texts** : dictionary containing TextBlock objects used throughout the mdf
 - **channels** : list of dictionaries that contain TextBlock objects related to each channel
 - * **name_addr** : channel name
 - * **comment_addr** : channel comment
 - **channel_group** : list of dictionaries that contain TextBlock objects related to each channel group
 - * **acq_name_addr** : channel group acquisition comment
 - * **comment_addr** : channel group comment
 - **conversion_tab** : list of dictionaries that contain TextBlock objects related to TABX and RTABX channel conversions
 - * **text_{n}** : n-th text of the VTABR conversion
 - * **default_addr** : default text
 - **conversions** : list of dictionaries that contain TextBlock objects related to channel conversions
 - * **name_addr** : conversions name
 - * **unit_addr** : channel unit_addr
 - * **comment_addr** : conversion comment
 - * **formula_addr** : formula text; only valid for algebraic conversions
 - **sources** : list of dictionaries that contain TextBlock objects related to channel sources
 - * **name_addr** : source name
 - * **path_addr** : source path_addr

* `comment_addr` : source comment

The `file_history` attribute is a list of (FileHistory, TextBlock) pairs .

The `channel_db` attribute is a dictionary that holds the (*data group index*, *channel index*) pair for all signals. This is used to speed up the `get_signal_by_name` method.

The `master_db` attribute is a dictionary that holds the *channel index* of the master channel for all data groups. This is used to speed up the `get_signal_by_name` method.

API

class `asammdf.mdf4.MDF4` (*name=None*, *memory='full'*, *version='4.10'*)

If the *name* exist it will be memorised otherwise an empty file will be created that can be later saved to disk

Parameters *name* : string

mdf file name

memory : str

memory optimization option; default *full*

- if *full* the data group binary data block will be memorised in RAM
- **if *low* the channel data is read from disk on request, and the metadata is** memorized into RAM
- if *minimum* only minimal data is memorized into RAM

version : string

mdf file version ('4.00', '4.10', '4.11'); default '4.10'

Attributes

name	(string) mdf file name
groups	(list) list of data groups
header	(HeaderBlock) mdf file header
file_history	(list) list of (FileHistory, TextBlock) pairs
comment	(TextBlock) mdf file comment
identification	(FileIdentificationBlock) mdf file start block
memory	(str) memory optimization option
version	(str) mdf version
channels_db	(dict) used for fast channel access by name; for each name key the value is a list of (group index, channel index) tuples
masters_db	(dict) used for fast master channel access; for each group index key the value is the master channel index

append (*signals*, *source_info='Python'*, *common_timebase=False*)

Appends a new data group.

For channel dependencies type Signals, the *samples* attribute must be a `numpy.recarray`

Parameters *signals* : list

list on *Signal* objects

source_info : str

source information; default 'Python'

common_timebase : bool

flag to hint that the signals have the same timebase

Examples

```
>>> # case 1 conversion type None
>>> s1 = np.array([1, 2, 3, 4, 5])
>>> s2 = np.array([-1, -2, -3, -4, -5])
>>> s3 = np.array([0.1, 0.04, 0.09, 0.16, 0.25])
>>> t = np.array([0.001, 0.002, 0.003, 0.004, 0.005])
>>> names = ['Positive', 'Negative', 'Float']
>>> units = ['+', '-', '.f']
>>> info = {}
>>> s1 = Signal(samples=s1, timestamps=t, unit='+', name='Positive')
>>> s2 = Signal(samples=s2, timestamps=t, unit='-', name='Negative')
>>> s3 = Signal(samples=s3, timestamps=t, unit='flts', name='Floats')
>>> mdf = MDF3('new.mdf')
>>> mdf.append([s1, s2, s3], 'created by asammdf v1.1.0')
>>> # case 2: VTAB conversions from channels inside another file
>>> mdf1 = MDF3('in.mdf')
>>> ch1 = mdf1.get("Channel1_VTAB")
>>> ch2 = mdf1.get("Channel2_VTABR")
>>> sigs = [ch1, ch2]
>>> mdf2 = MDF3('out.mdf')
>>> mdf2.append(sigs, 'created by asammdf v1.1.0')
```

attach(data, file_name=None, comment=None, compression=True, mime='application/octet-stream')

attach embedded attachment as application/octet-stream

Parameters **data** : bytes

data to be attached

file_name : str

string file name

comment : str

attachment comment

compression : bool

use compression for embedded attachment data

mime : str

mime type string

close()

if the MDF was created with memory=False and new channels have been appended, then this must be called just before the object is not used anymore to clean-up the temporary file

extract_attachment(index)

extract attachment *index* data. If it is an embedded attachment, then this method creates the new file according to the attachment file name information

Parameters `index` : int

attachment index

Returns `data` : bytes | str

attachment data

get (`name=None, group=None, index=None, raster=None, samples_only=False, data=None`)

Gets channel samples. Channel can be specified in two ways:

- using the first positional argument `name`
 - **if there are multiple occurrences for this channel then the `group` and `index` arguments can be used to select a specific group.**
 - **if there are multiple occurrences for this channel and either the `group` or `index` arguments is `None` then a warning is issued**
- **using the group number (keyword argument `group`) and the channel number (keyword argument `index`).** Use `info` method for group and channel numbers

If the `raster` keyword argument is not `None` the output is interpolated accordingly

Parameters `name` : string

name of channel

group : int

0-based group index

index : int

0-based channel index

raster : float

time raster in seconds

samples_only : bool

if `True` return only the channel samples as numpy array; if `False` return a `Signal` object

Returns `res` : (numpy.array | `Signal`)

returns `Signal` if `samples_only*=False` (default option), otherwise returns numpy.array The `Signal` samples are:

- **numpy recarray for channels that have composition/channel array address or for channel of type `BYTEARRAY`, `CANOPENDATE`, `CANOPENTIME`**
- numpy array for all the rest

Raises `MdfError` :

- * **if the channel name is not found**
- * **if the group index is out of range**
- * **if the channel index is out of range**

get_channel_comment (`name=None, group=None, index=None`)

Gets channel comment.

Channel can be specified in two ways:

- using the first positional argument *name*
 - if there are multiple occurrences for this channel then the *group* and *index* arguments can be used to select a specific group.
 - if there are multiple occurrences for this channel and either the *group* or *index* arguments is *None* then a warning is issued
- using the group number (keyword argument *group*) and the channel number (keyword argument *index*). Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters *name* : string

name of channel

group : int

0-based group index

index : int

0-based channel index

Returns *comment* : str

found channel comment

get_channel_unit (*name=None, group=None, index=None*)

Gets channel unit.

Channel can be specified in two ways:

- using the first positional argument *name*
 - if there are multiple occurrences for this channel then the *group* and *index* arguments can be used to select a specific group.
 - if there are multiple occurrences for this channel and either the *group* or *index* arguments is *None* then a warning is issued
- using the group number (keyword argument *group*) and the channel number (keyword argument *index*). Use *info* method for group and channel numbers

If the *raster* keyword argument is not *None* the output is interpolated accordingly.

Parameters *name* : string

name of channel

group : int

0-based group index

index : int

0-based channel index

Returns *unit* : str

found channel unit

get_master (*index, data=None*)

returns master channel samples for given group

Parameters *index* : int

group index

data : bytes

data block raw bytes; default None

Returns **t** : numpy.array

master channel samples

info()

get MDF information as a dict

Examples

```
>>> mdf = MDF4('test.mdf')
>>> mdf.info()
```

save (*dst*="", *overwrite*=None, *compression*=0)

Save MDF to *dst*. If *dst* is not provided the the destination file name is the MDF name. If *overwrite* is *True* then the destination file is overwritten, otherwise the file name is appened with '*_<cntr>*', were '*<cntr>*' is the first conter that produces a new file name (that does not already exist in the filesystem)

Parameters **dst** : str

destination file name, Default ''

overwrite : bool

overwrite flag, default *False*

compression : int

use compressed data blocks, default 0; valid since version 4.10

- 0 - no compression
- 1 - deflate (slower, but produces smaller files)
- 2 - transposition + deflate (slowest, but produces the smallest files)

MDF version 4 blocks

The following classes implement different MDF version3 blocks.

AttachmentBlock Class

class `asammdf.mdf4.AttachmentBlock` (***kargs*)

ATBLOCK class

When adding new attachments only embedded attachemnts are allowed, with keyword argument *data* of type bytes

Channel Class

class `asammdf.mdf4.Channel` (***kargs*)

CNBLOCK class

ChannelConversion Class

```
class asammdf.mdf4.ChannelConversion(**kargs)
    CCBLOCK class
```

ChannelGroup Class

```
class asammdf.mdf4.ChannelGroup(**kargs)
    CGBLOCK class
```

DataGroup Class

```
class asammdf.mdf4.DataGroup(**kargs)
    DGBLOCK class
```

DataList Class

```
class asammdf.mdf4.DataList(**kargs)
    DLBLOCK class
```

DataBlock Class

```
class asammdf.mdf4.DataBlock(**kargs)
    DTBLOCK class

    Parameters address : int
                    DTBLOCK address inside the file

    stream : int
                    file handle
```

FileIdentificationBlock Class

```
class asammdf.mdf4.FileIdentificationBlock(**kargs)
    IDBLOCK class
```

HeaderBlock Class

```
class asammdf.mdf4.HeaderBlock(**kargs)
    HDBLOCK class
```

SourceInformation Class

```
class asammdf.mdf4.SourceInformation(**kargs)
    SIBLOCK class
```

FileHistory Class

```
class asammdf.mdf4.FileHistory (**kargs)
    FHBLOCK class
```

TextBlock Class

```
class asammdf.mdf4.TextBlock (**kargs)
    common TXBLOCK and MDBLOCK class
```

6.2.2 Notes about *memory* argument

By default when the *MDF* object is created all data is loaded into RAM (*memory=full*). This will give you the best performance from *asammdf*.

However if you reach the physical memory limit *asammdf* gives you two options:

- *memory=low* : only the metadata is loaded into RAM, the raw channel data is loaded when needed
- *memory=minimum* : only minimal data is loaded into RAM.

MDF created with *memory='full'*

Advantages

- best performance

Disadvantages

- higher RAM usage, there is the chance the file will exceed available RAM

Use case

- when data fits inside the system RAM

MDF created with *memory='low'*

Advantages

- lower RAM usage than *memory=full*
- can handle files that do not fit in the available physical memory

Disadvantages

- slow performance for getting channel data
- must call *close* method to release the temporary file used in case of appending.

Note: it is advised to use the *MDF* context manager in this case

Use case

- when *default* data exceeds available RAM
- it is advised to avoid getting individual channels when using this option.

Instead you can get performance close to memory='full' if you use the *select* method with the list of target channels.

Note: See benchmarks for the effects of using the flag

MDF created with memory='minimum'

Advantages

- lowest RAM usage
- can handle files that do not fit in the available physical memory
- handle big files on 32 bit Python

Disadvantages

- slightly slower performance compared to memory=low
- must call *close* method to release the temporary file used in case of appending.

Note: See benchmarks for the effects of using the flag

6.3 Signal

class `asammdf.signal.Signal` (*samples=None, timestamps=None, unit="", name="", info=None, comment=""*)

The `Signal` represents a signal described by its samples and timestamps. It can do arithmetic operations against other `Signal` or numeric type. The operations are computed in respect to the timestamps (time correct). The integer signals are not interpolated, instead the last value relative to the current timestamp is used. *samples*, *timestamps* and *name* are mandatory arguments.

Parameters **samples** : `numpy.array` | `list` | `tuple`

signal samples

timestamps : `numpy.array` | `list` | `tuple`

signal timestamps

unit : `str`

signal unit

name : `str`

signal name

info : `dict`

dict that contains extra information about the signal , default *None*

comment : `str`

signal comment, default ""

astype (*np_type*)

returns new *Signal* with samples of dtype *np_type*

cut (*start=None, stop=None*)

Cuts the signal according to the *start* and *stop* values, by using the insertion indexes in the signal's *time* axis.

Parameters *start* : float

start timestamp for cutting

stop : float

stop timestamp for cutting

Returns *result* : Signal

new *Signal* cut from the original

Examples

```
>>> new_sig = old_sig.cut(1.0, 10.5)
>>> new_sig.timestamps[0], new_sig.timestamps[-1]
0.98, 10.48
```

extend (*other*)

extend signal with samples from another signal

Parameters *other* : Signal

interp (*new_timestamps*)

returns a new *Signal* interpolated using the *new_timestamps*

plot ()

plot Signal samples

6.4 Examples

6.4.1 Working with MDF

```
from __future__ import print_function, division
from asammdf import MDF, Signal, configure
import numpy as np

# configure asammdf to optimize disk space usage
configure(integer_compacting=True)
# configure asammdf to split data blocks on 10KB blocks
configure(split_data_blocks=True, split_threshold=10*1024)

# create 3 Signal objects

timestamps = np.array([0.1, 0.2, 0.3, 0.4, 0.5], dtype=np.float32)

# uint8
s_uint8 = Signal(samples=np.array([0, 1, 2, 3, 4], dtype=np.uint8),
                 timestamps=timestamps,
                 name='UInt8_Signal',
                 unit='u1')

# int32
```

```
s_int32 = Signal(samples=np.array([-20, -10, 0, 10, 20], dtype=np.int32),
                 timestamps=timestamps,
                 name='Int32_Signal',
                 unit='i4')

# float64
s_float64 = Signal(samples=np.array([-20, -10, 0, 10, 20], dtype=np.float64),
                  timestamps=timestamps,
                  name='Float64_Signal',
                  unit='f8')

# create empty Mdf version 4.00 file
mdf4 = MDF(version='4.10')

# append the 3 signals to the new file
signals = [s_uint8, s_int32, s_float64]
mdf4.append(signals, 'Created by Python')

# save new file
mdf4.save('my_new_file.mf4', overwrite=True)

# convert new file to mdf version 3.10 with lower possible RAM usage
mdf3 = mdf4.convert(to='3.10', memory='minimum')
print(mdf3.version)

# get the float signal
sig = mdf3.get('Float64_Signal')
print(sig)

# cut measurement from 0.3s to end of measurement
mdf4_cut = mdf4.cut(start=0.3)
mdf4_cut.get('Float64_Signal').plot()

# cut measurement from start of measurement to 0.4s
mdf4_cut = mdf4.cut(stop=0.45)
mdf4_cut.get('Float64_Signal').plot()

# filter some signals from the file
mdf4 = mdf4.filter(['Int32_Signal', 'Uint8_Signal'])

# save using zipped transpose deflate blocks
mdf4.save('out.mf4', compression=2, overwrite=True)
```

6.4.2 Working with Signal

```
from __future__ import print_function, division
from asammdf import Signal
import numpy as np

# create 3 Signal objects with different time stamps

# uint8 with 100ms time raster
timestamps = np.array([0.1 * t for t in range(5)], dtype=np.float32)
s_uint8 = Signal(samples=np.array([t for t in range(5)], dtype=np.uint8),
                 timestamps=timestamps,
```

```

        name='Uint8_Signal',
        unit='u1')

# int32 with 50ms time raster
timestamps = np.array([0.05 * t for t in range(10)], dtype=np.float32)
s_int32 = Signal(samples=np.array(list(range(-500, 500, 100))), dtype=np.int32),
                 timestamps=timestamps,
                 name='Int32_Signal',
                 unit='i4')

# float64 with 300ms time raster
timestamps = np.array([0.3 * t for t in range(3)], dtype=np.float32)
s_float64 = Signal(samples=np.array(list(range(2000, -1000, -1000))), dtype=np.int32),
                  timestamps=timestamps,
                  name='Float64_Signal',
                  unit='f8')

# map signals
xs = np.linspace(-1, 1, 50)
ys = np.linspace(-1, 1, 50)
X, Y = np.meshgrid(xs, ys)
vals = np.linspace(0, 180. / np.pi, 100)
phi = np.ones((len(vals), 50, 50), dtype=np.float64)
for i, val in enumerate(vals):
    phi[i] *= val
R = 1 - np.sqrt(X**2 + Y**2)
samples = np.cos(2 * np.pi * X + phi) * R
print(phi.shape, samples.shape)
timestamps = np.arange(0, 2, 0.02)

s_map = Signal(samples=samples,
               timestamps=timestamps,
               name='Variable Map Signal',
               unit='dB')
s_map.plot()

prod = s_float64 * s_uint8
prod.name = 'Uint8_Signal * Float64_Signal'
prod.unit = '*'
prod.plot()

pow2 = s_uint8 ** 2
pow2.name = 'Uint8_Signal ^ 2'
pow2.unit = 'u1^2'
pow2.plot()

allsum = s_uint8 + s_int32 + s_float64
allsum.name = 'Uint8_Signal + Int32_Signal + Float64_Signal'
allsum.unit = '+'
allsum.plot()

# inplace operations
pow2 *= -1
pow2.name = '- Uint8_Signal ^ 2'
pow2.plot()

# cut signal

```

```
s_int32.plot()  
cut_signal = s_int32.cut(start=0.2, stop=0.35)  
cut_signal.plot()
```


asammdf relies heavily on *dict* objects. Starting with Python 3.6 the *dict* objects are more compact and ordered (implementation detail); *asammdf* uses takes advantage of those changes so for best performance it is advised to use Python ≥ 3.6 .

7.1 Intro

The benchmarks were done using two test files (available here <https://github.com/danielhrisca/asammdf/issues/14>) (for mdf version 3 and 4) of around 170MB. The files contain 183 data groups and a total of 36424 channels.

asammdf 2.8.1 was compared against *mdfreader* 2.7.3 (latest versions from PyPI). *mdfreader* seems to be the most used Python package to handle MDF files, and it also supports both version 3 and 4 of the standard.

The three benchmark categories are file open, file save and extracting the data for all channels inside the file(36424 calls). For each category two aspects were noted: elapsed time and peak RAM usage.

7.2 Dependencies

You will need the following packages to be able to run the benchmark script

- psutil
- mdfreader

7.3 Usage

Extract the test files from the archive, or provide a folder that contains the files “test.mdf” and “test.mf4”. Run the module *bench.py* (see `-help` option for available options)

7.4 x64 Python results

Benchmark environment

- 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
- Windows-10-10.0.16299-SP0
- Intel64 Family 6 Model 69 Stepping 1, GenuineIntel
- 16GB installed RAM

Notations used in the results

- full = asammdf MDF object created with memory=full (everything loaded into RAM)
- low = asammdf MDF object created with memory=low (raw channel data not loaded into RAM, but metadata loaded to RAM)
- minimum = asammdf MDF object created with memory=full (lowest possible RAM usage)
- compress = mdfreader mdf object created with compression=blosc
- noDataLoading = mdfreader mdf object read with noDataLoading=True

Files used for benchmark:

- 183 groups
- 36424 channels

7.4.1 Raw data

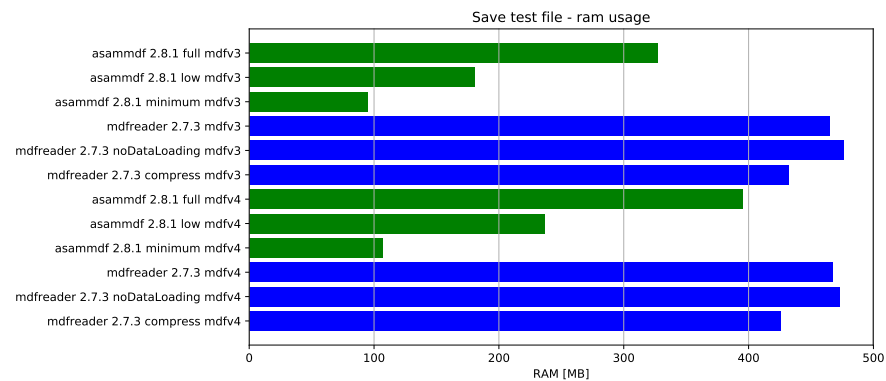
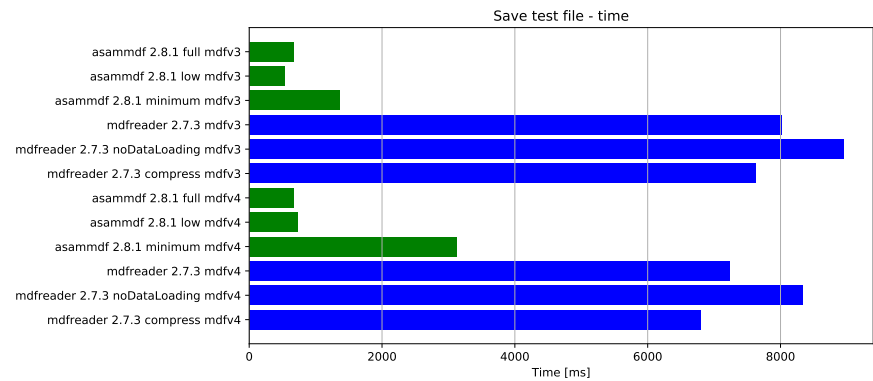
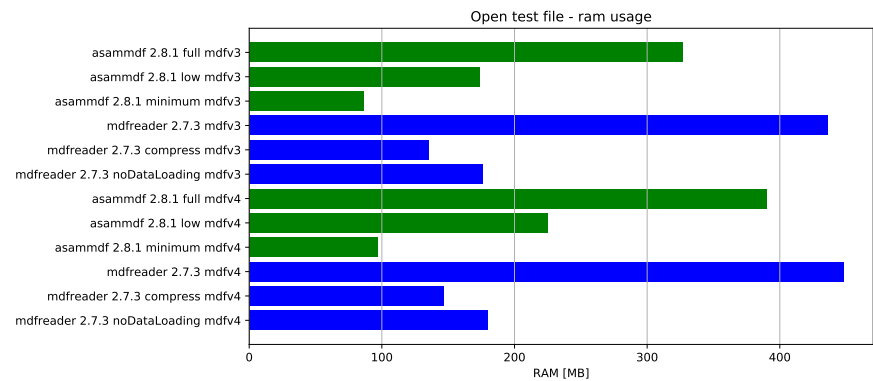
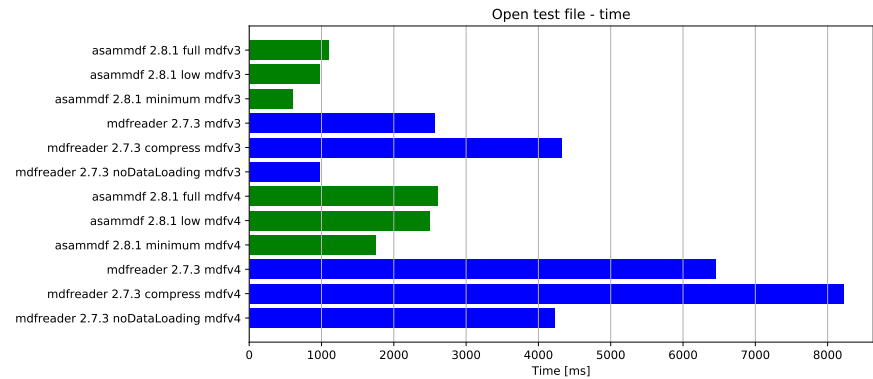
Open file	Time [ms]	RAM [MB]
asammdf 2.8.1 full mdfv3	1100	327
asammdf 2.8.1 low mdfv3	980	174
asammdf 2.8.1 minimum mdfv3	599	86
mdfreader 2.7.3 mdfv3	2567	436
mdfreader 2.7.3 compress mdfv3	4324	135
mdfreader 2.7.3 noDataLoading mdfv3	973	176
asammdf 2.8.1 full mdfv4	2613	390
asammdf 2.8.1 low mdfv4	2491	225
asammdf 2.8.1 minimum mdfv4	1749	97
mdfreader 2.7.3 mdfv4	6457	448
mdfreader 2.7.3 compress mdfv4	8219	147
mdfreader 2.7.3 noDataLoading mdfv4	4221	180

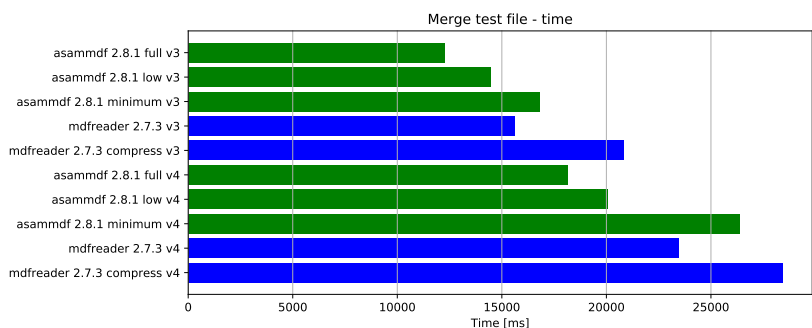
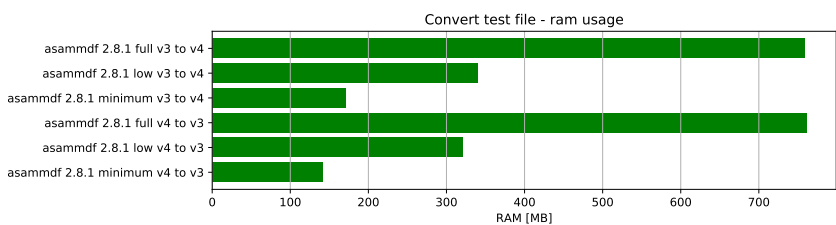
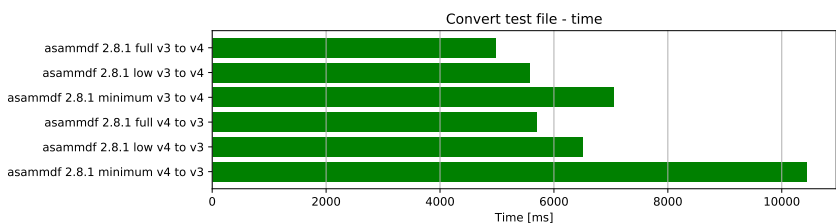
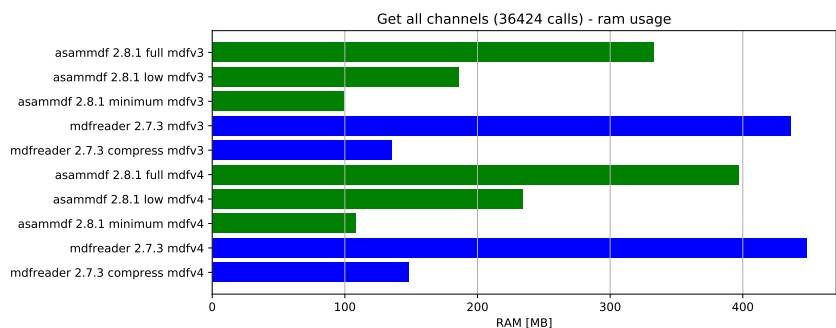
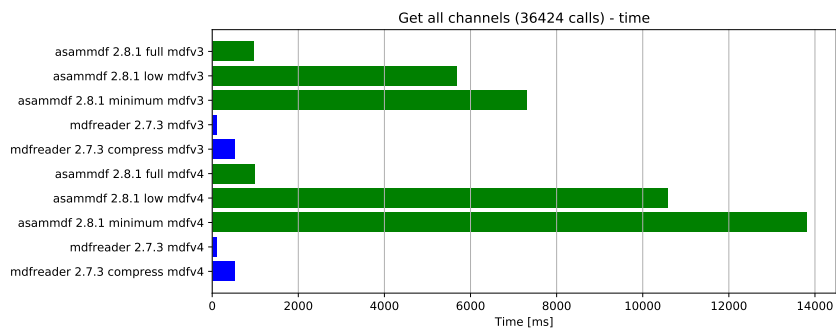
Save file	Time [ms]	RAM [MB]
asammdf 2.8.1 full mdfv3	676	327
asammdf 2.8.1 low mdfv3	541	181
asammdf 2.8.1 minimum mdfv3	1363	95
mdfreader 2.7.3 mdfv3	8013	465
mdfreader 2.7.3 noDataLoading mdfv3	8948	476
mdfreader 2.7.3 compress mdfv3	7629	432
asammdf 2.8.1 full mdfv4	672	395
asammdf 2.8.1 low mdfv4	736	237
asammdf 2.8.1 minimum mdfv4	3127	107
mdfreader 2.7.3 mdfv4	7237	467
mdfreader 2.7.3 noDataLoading mdfv4	8332	473
mdfreader 2.7.3 compress mdfv4	6791	426

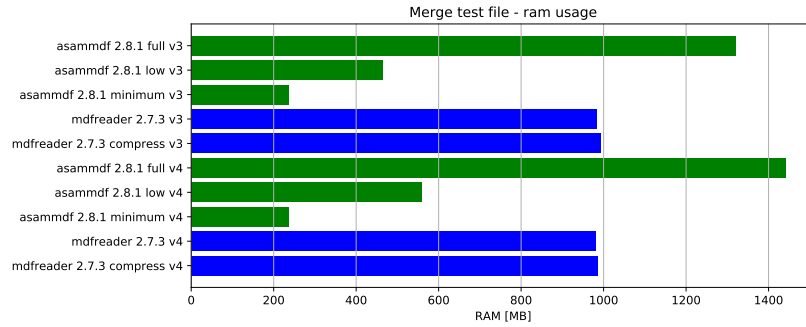
Get all channels (36424 calls)	Time [ms]	RAM [MB]
asammdf 2.8.1 full mdfv3	967	333
asammdf 2.8.1 low mdfv3	5690	186
asammdf 2.8.1 minimum mdfv3	7296	99
mdfreader 2.7.3 mdfv3	95	436
mdfreader 2.7.3 compress mdfv3	531	135
asammdf 2.8.1 full mdfv4	988	397
asammdf 2.8.1 low mdfv4	10572	234
asammdf 2.8.1 minimum mdfv4	13803	108
mdfreader 2.7.3 mdfv4	95	448
mdfreader 2.7.3 compress mdfv4	534	148

Convert file	Time [ms]	RAM [MB]
asammdf 2.8.1 full v3 to v4	4986	759
asammdf 2.8.1 low v3 to v4	5573	340
asammdf 2.8.1 minimum v3 to v4	7049	171
asammdf 2.8.1 full v4 to v3	5705	761
asammdf 2.8.1 low v4 to v3	6510	321
asammdf 2.8.1 minimum v4 to v3	10434	142

Merge files	Time [ms]	RAM [MB]
asammdf 2.8.1 full v3	12251	1320
asammdf 2.8.1 low v3	14453	464
asammdf 2.8.1 minimum v3	16830	236
mdfreader 2.7.3 v3	15635	983
mdfreader 2.7.3 compress v3	20812	993
asammdf 2.8.1 full v4	18172	1441
asammdf 2.8.1 low v4	20083	558
asammdf 2.8.1 minimum v4	26374	237
mdfreader 2.7.3 v4	23450	981
mdfreader 2.7.3 compress v4	28421	985







7.4.2 Graphical results

7.5 x86 Python results

Benchmark environment

- 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
- Windows-10-10.0.16299-SP0
- Intel64 Family 6 Model 69 Stepping 1, GenuineIntel
- 16GB installed RAM

Notations used in the results

- full = asammdf MDF object created with memory=full (everything loaded into RAM)
- low = asammdf MDF object created with memory=low (raw channel data not loaded into RAM, but metadata loaded to RAM)
- minimum = asammdf MDF object created with memory=full (lowest possible RAM usage)
- compress = mdfreader mdm object created with compression=blosc
- noDataLoading = mdfreader mdm object read with noDataLoading=True

Files used for benchmark:

- 183 groups
- 36424 channels

7.5.1 Raw data

Open file	Time [ms]	RAM [MB]
asammdf 2.8.1 full mdfv3	1259	260
asammdf 2.8.1 low mdfv3	1076	106
asammdf 2.8.1 minimum mdfv3	767	52
mdfreader 2.7.3 mdfv3	3146	392
mdfreader 2.7.3 noDataLoading mdfv3	1159	102
asammdf 2.8.1 full mdfv4	2792	299
asammdf 2.8.1 low mdfv4	2645	133
asammdf 2.8.1 minimum mdfv4	2070	58
mdfreader 2.7.3 mdfv4	7372	397
mdfreader 2.7.3 noDataLoading mdfv4	4526	104

Save file	Time [ms]	RAM [MB]
asammdf 2.8.1 full mdfv3	581	263
asammdf 2.8.1 low mdfv3	688	114
asammdf 2.8.1 minimum mdfv3	1931	58
mdfreader 2.7.3 mdfv3	8902	412
mdfreader 2.7.3 noDataLoading mdfv3	10490	420
asammdf 2.8.1 full mdfv4	843	303
asammdf 2.8.1 low mdfv4	959	143
asammdf 2.8.1 minimum mdfv4	3698	67
mdfreader 2.7.3 mdfv4	8084	417
mdfreader 2.7.3 noDataLoading mdfv4	9524	426

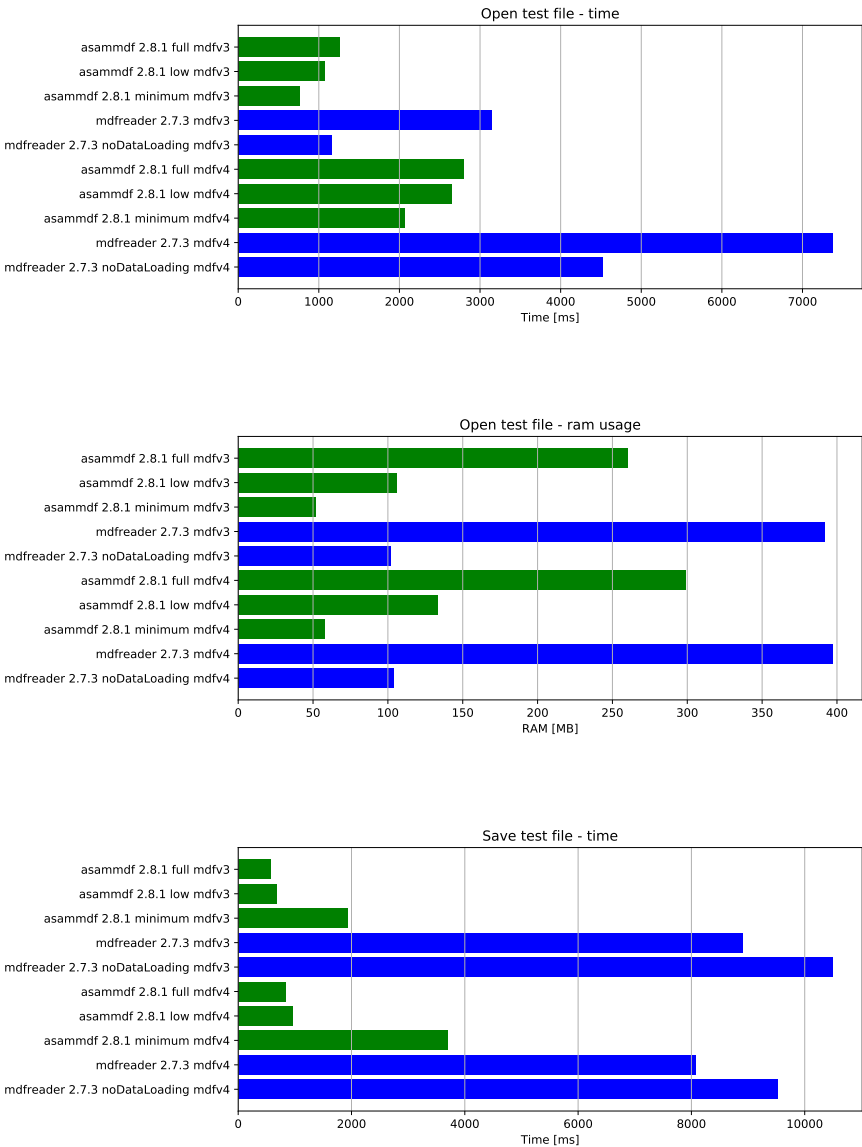
Get all channels (36424 calls)	Time [ms]	RAM [MB]
asammdf 2.8.1 full mdfv3	1278	265
asammdf 2.8.1 low mdfv3	18354	116
asammdf 2.8.1 minimum mdfv3	19288	63
mdfreader 2.7.3 mdfv3	117	392
asammdf 2.8.1 full mdfv4	1266	303
asammdf 2.8.1 low mdfv4	20515	141
asammdf 2.8.1 minimum mdfv4	23939	65
mdfreader 2.7.3 mdfv4	116	398

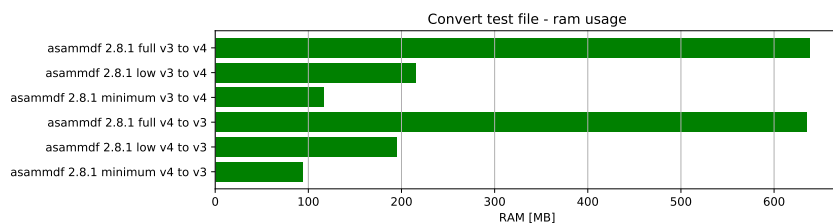
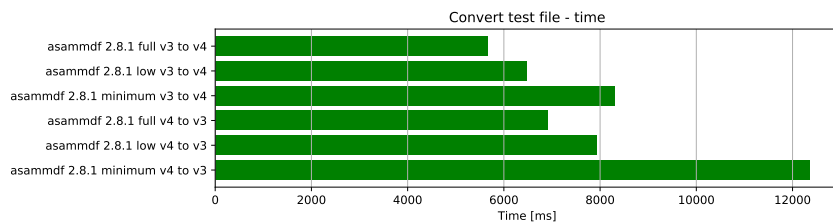
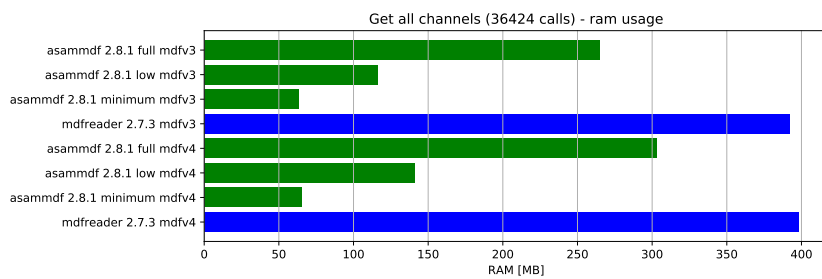
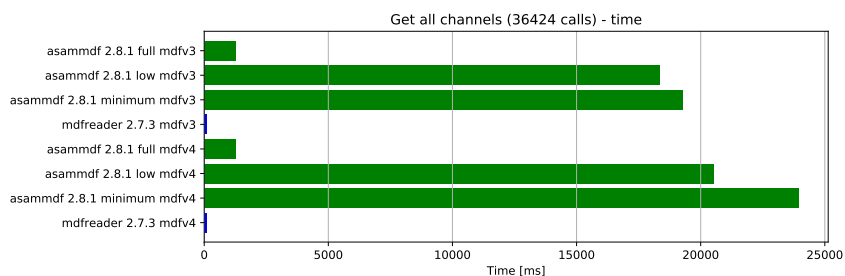
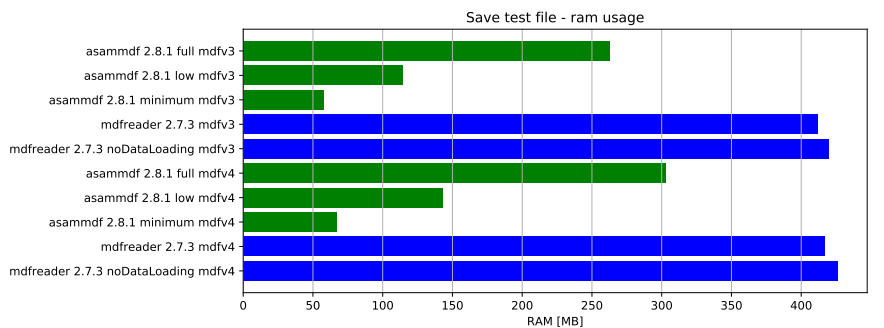
Convert file	Time [ms]	RAM [MB]
asammdf 2.8.1 full v3 to v4	5667	638
asammdf 2.8.1 low v3 to v4	6483	215
asammdf 2.8.1 minimum v3 to v4	8301	117
asammdf 2.8.1 full v4 to v3	6910	635
asammdf 2.8.1 low v4 to v3	7938	195
asammdf 2.8.1 minimum v4 to v3	12352	94

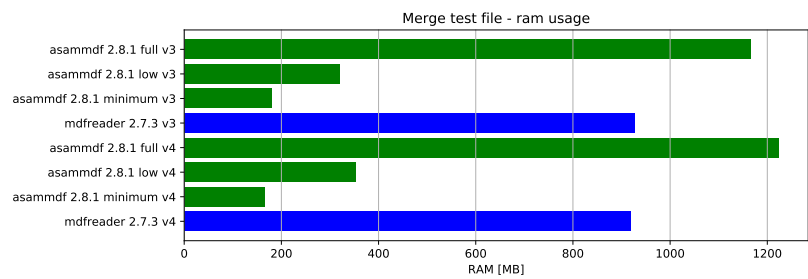
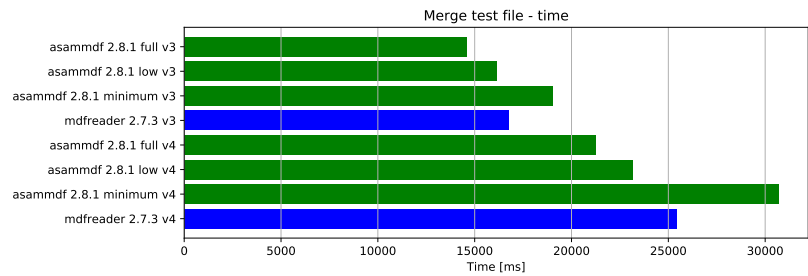
Merge files	Time [ms]	RAM [MB]
asammdf 2.8.1 full v3	14564	1165
asammdf 2.8.1 low v3	16148	319
asammdf 2.8.1 minimum v3	19046	180
mdfreader 2.7.3 v3	16765	928
asammdf 2.8.1 full v4	21262	1223
asammdf 2.8.1 low v4	23150	352
asammdf 2.8.1 minimum v4	30687	166
mdfreader 2.7.3 v4	25437	919

- mdfreader got a MemoryError

7.5.2 Graphical results







CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

A

`astype()` (`asammdf.signal.Signal` method), 56
`AttachmentBlock` (class in `asammdf.mdf4`), 53

C

`Channel` (class in `asammdf.mdf2`), 21
`Channel` (class in `asammdf.mdf3`), 37
`Channel` (class in `asammdf.mdf4`), 53
`ChannelConversion` (class in `asammdf.mdf2`), 23
`ChannelConversion` (class in `asammdf.mdf3`), 39
`ChannelConversion` (class in `asammdf.mdf4`), 54
`ChannelDependency` (class in `asammdf.mdf2`), 24
`ChannelDependency` (class in `asammdf.mdf3`), 40
`ChannelExtension` (class in `asammdf.mdf2`), 25
`ChannelExtension` (class in `asammdf.mdf3`), 41
`ChannelGroup` (class in `asammdf.mdf2`), 26
`ChannelGroup` (class in `asammdf.mdf3`), 42
`ChannelGroup` (class in `asammdf.mdf4`), 54
`configure()` (in module `asammdf`), 13
`convert()` (`asammdf.mdf.MDF` method), 14
`cut()` (`asammdf.mdf.MDF` method), 14
`cut()` (`asammdf.signal.Signal` method), 56

D

`DataBlock` (class in `asammdf.mdf4`), 54
`DataGroup` (class in `asammdf.mdf2`), 27
`DataGroup` (class in `asammdf.mdf3`), 43
`DataGroup` (class in `asammdf.mdf4`), 54
`DataList` (class in `asammdf.mdf4`), 54

E

`export()` (`asammdf.mdf.MDF` method), 15
`extend()` (`asammdf.signal.Signal` method), 57

F

`FileHistory` (class in `asammdf.mdf4`), 55
`FileIdentificationBlock` (class in `asammdf.mdf2`), 28
`FileIdentificationBlock` (class in `asammdf.mdf3`), 44
`FileIdentificationBlock` (class in `asammdf.mdf4`), 54

`filter()` (`asammdf.mdf.MDF` method), 15

H

`HeaderBlock` (class in `asammdf.mdf2`), 29
`HeaderBlock` (class in `asammdf.mdf3`), 44
`HeaderBlock` (class in `asammdf.mdf4`), 54

I

`interp()` (`asammdf.signal.Signal` method), 57
`iter_to_pandas()` (`asammdf.mdf.MDF` method), 15

M

`MDF` (class in `asammdf.mdf`), 14
`merge()` (`asammdf.mdf.MDF` static method), 15

P

`plot()` (`asammdf.signal.Signal` method), 57
`ProgramBlock` (class in `asammdf.v2blocks`), 29
`ProgramBlock` (class in `asammdf.v3blocks`), 45

R

`resample()` (`asammdf.mdf.MDF` method), 16

S

`SampleReduction` (class in `asammdf.v2blocks`), 30
`SampleReduction` (class in `asammdf.v3blocks`), 46
`select()` (`asammdf.mdf.MDF` method), 16
`Signal` (class in `asammdf.signal`), 56
`SourceInformation` (class in `asammdf.mdf4`), 54

T

`TextBlock` (class in `asammdf.mdf2`), 31
`TextBlock` (class in `asammdf.mdf3`), 46
`TextBlock` (class in `asammdf.mdf4`), 55
`TriggerBlock` (class in `asammdf.mdf2`), 31
`TriggerBlock` (class in `asammdf.mdf3`), 47