
asammdf Documentation

Release 2.5.3

Daniel Hrisca

Sep 27, 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	49
8	Indices and tables	63

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

asammdf supports both MDF version 3 and 4 formats.

asammdf works on Python 2.7, and Python ≥ 3.4

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

- read sorted and unsorted MDF v3 and v4 files
- files are loaded in RAM for fast operations
- handle large files (exceeding the available RAM) using *load_measured_data = False* 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
- remove data group by index or by specifying a channel name inside the target data group
- append new channels
- filter a subset of channels from original mdf file
- cut measurement to specified time interval
- convert to different mdf version
- export to Excel, HDF5 and CSV
- merge multiple files sharing the same internal structure
- add and extract attachments
- mdf 4.10 zipped blocks

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)
 - mdf 4 channel arrays
 - xml schema for TXBLOCK and MDBLOCK

CHAPTER 4

Dependencies

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

optional dependencies needed for exports

- pandas : for DataFrame export
- h5py : for HDF5 export
- xlswriter : for Excel export

CHAPTER 5

Installation

asammdf is available on

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

```
pip install asammdf
```


MDF

This class acts as a proxy for the MDF3 and MDF4 classes. All attribute access is delegated to the underlying *_file* 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, load_measured_data=True, version='3.20'*)
Unified access to MDF v3 and v4 files.

Parameters **name** : string

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

load_measured_data : bool

load data option; default *True*

- if *True* the data group binary data block will be loaded in RAM
- if *False* the channel data is read from disk on request

version : string

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

Methods

convert
cut
export
filter
iter_to_pandas
merge

convert (*to*, *load_measured_data=True*)
 convert MDF to other versions

Parameters to : str

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

load_measured_data : bool

load data option; default *True*

- if *True* the data group binary data block will be loaded in RAM
- if *False* the channel data is stored to a temporary file and read from disk on request

Returns out : MDF

new MDF object

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

Returns out : MDF

new MDF object

export (*format*, *filename=None*)
 export MDF to other formats. The *MDF* file name is used is available, else the *filename* aragument must be provided.

Parameters format : string

can be one of the following:

- *csv* : CSV export that uses the ";" delimiter. This option wil generate a new csv file for each data group (<MDFNAME>_DataGroup_XX.csv).
- *hdf5* : HDF5 file output; each *MDF* data group is mapped to a *HDF5* group with the name 'DataGroup_xx' (where xx is the index)
- *excel* : Excel file output (very slow). This option wil generate a new excel file for each data group (<MDFNAME>_DataGroup_XX.xlsx).

filename : string

export file name

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

Parameters channels : list

list of channel names to be filtered

Returns mdf : MDF

new MDF file

iter_to_pandas ()

generator that yields channel groups as pandas DataFrames

static merge (*files*, *load_measured_data=True*)

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

load_measured_data : bool

load data option; default *True*

- if *True* the data group binary data block will be loaded in RAM
- if *False* the channel data is stored to a temporary file and read from disk on request

Returns merged : MDF

new MDF object with merged channels

Raises MdfException : if there are inconsistencies between the files

merged MDF object

MDF3 and MDF4 classes

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 `load_measured_data = False`
- `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*, *load_measured_data=True*, *version='3.20'*)

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

load_measured_data : bool

load data option; default *True*

- if *True* the data group binary data block will be loaded in RAM
- if *False* the channel data is read from disk on request

version : string

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

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
load_measured_data	(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

Methods

add_trigger
append
close
get
info
iter_get_triggers
remove
save

add_trigger (*group, time, pre_time=0, post_time=0, comment=''*)

add trigger to data group

Parameters **group** : int

group index

time : 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 `load_measured_data=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

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

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

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

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

remove (*group=None, name=None*)
Remove data group. Use *group* or *name* keyword arguments to identify the group's index. *group* has priority

Parameters name : string

name of the channel inside the data group to be removed

group : int

data group index to be removed

Examples

```
>>> mdf = MDF3('test.mdf')
>>> mdf.remove(group=3)
>>> mdf.remove(name='VehicleSpeed')
```

save (*dst='', overwrite=False, 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 '_xx', were 'xx' 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 *file_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-dependent 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 `file_stream` : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     ch1 = Channel(file_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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

ChannelConversion Class

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

CCBLOCK class derived from *dict*

The ChannelConversion object can be created in two modes:

- using the *file_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 `file_stream` : file handle

mdf file handle

address : int

block address inside mdf file

Examples

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

Attributes

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

ChannelDependency Class

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

CDBLOCK class derived from *dict*

Currently the ChannelDependency object can only be created using the *file_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)
- data - 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 `file_stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

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

Methods

<code>clear</code>	
<code>copy</code>	Generic (shallow and deep) copying operations.
<code>fromkeys</code>	
<code>get</code>	
<code>items</code>	
<code>keys</code>	
<code>pop</code>	
<code>popitem</code>	
<code>setdefault</code>	
<code>update</code>	
<code>values</code>	

ChannelExtension Class

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

CEBLOCK class derived from *dict*

The ChannelExtension object can be created in two modes:

- using the *file_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 `file_stream` : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

ChannelGroup Class

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

CGBLOCK class derived from *dict*

The ChannelGroup object can be created in two modes:

- using the *file_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 **file_stream** : file handle

mdf file handle

address : int

block address inside mdf file

Examples

```
>>> with open('test.mdf', 'rb') as mdf:
...     cg1 = ChannelGroup(file_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
----------------	-------------------------------------

Methods

clear

copy

Generic (shallow and deep) copying operations.

fromkeys

get

items

keys

Continued on next page

Table 6.7 – continued from previous page

pop
popitem
setdefault
update
values

DataGroup Class

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

DGBLOCK class derived from *dict*

The DataGroup object can be created in two modes:

- using the *file_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 *file_stream* : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
Continued on next page	

Table 6.8 – continued from previous page

keys
pop
popitem
setdefault
update
values

FileIdentificationBlock Class

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

IDBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the *file_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 *file_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
----------------	---

Methods

clear

Continued on next page

Table 6.9 – continued from previous page

<code>copy</code>	Generic (shallow and deep) copying operations.
<code>fromkeys</code>	
<code>get</code>	
<code>items</code>	
<code>keys</code>	
<code>pop</code>	
<code>popitem</code>	
<code>setdefault</code>	
<code>update</code>	
<code>values</code>	

HeaderBlock Class

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

The TriggerBlock object can be created in two modes:

- using the *file_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 *file_stream* : file handle

mdf file handle

Attributes

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

ProgramBlock Class

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

PRBLOCK class derived from *dict*

The ProgramBlock object can be created in two modes:

- using the *file_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 *file_stream* : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

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

Methods

clear	
Continued on next page	

Table 6.11 – continued from previous page

<code>copy</code>	Generic (shallow and deep) copying operations.
<code>fromkeys</code>	
<code>get</code>	
<code>items</code>	
<code>keys</code>	
<code>pop</code>	
<code>popitem</code>	
<code>setdefault</code>	
<code>update</code>	
<code>values</code>	

SampleReduction Class

class `asammdf.mdf3.SampleReduction` (**kargs)
 SRBLOCK class derived from *dict*

Currently the SampleReduction object can only be created by using the *file_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 **file_stream** : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

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

Methods

<code>clear</code>	
<code>copy</code>	Generic (shallow and deep) copying operations.
<code>fromkeys</code>	
<code>get</code>	
<code>items</code>	
<code>keys</code>	
Continued on next page	

Table 6.12 – continued from previous page

pop
popitem
setdefault
update
values

TextBlock Class

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

TXBLOCK class derived from *dict*

The ProgramBlock object can be created in two modes:

- using the *file_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 **file_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

Methods

clear

Continued on next page

Table 6.13 – continued from previous page

<code>copy</code>	Generic (shallow and deep) copying operations.
<code>from_text</code>	
<code>fromkeys</code>	
<code>get</code>	
<code>items</code>	
<code>keys</code>	
<code>pop</code>	
<code>popitem</code>	
<code>setdefault</code>	
<code>update</code>	
<code>values</code>	

TriggerBlock Class

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

TRBLOCK class derived from *dict*

The TriggerBlock object can be created in two modes:

- using the *file_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 *file_stream* : file handle

mdf file handle

address : int

block address inside mdf file

Attributes

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

Methods

<code>clear</code>

Continued on next page

Table 6.14 – continued from previous page

<code>copy</code>	Generic (shallow and deep) copying operations.
<code>fromkeys</code>	
<code>get</code>	
<code>items</code>	
<code>keys</code>	
<code>pop</code>	
<code>popitem</code>	
<code>setdefault</code>	
<code>update</code>	
<code>values</code>	

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, load_measured_data=True, version='4.00'*)

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

load_measured_data : bool

load data option; default *True*

- if *True* the data group binary data block will be loaded in RAM
- if *False* the channel data is read from disk on request

version : string

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

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
load_measured_data	(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

Methods

`append`

`attach`

`close`

Continued on next page

Table 6.15 – continued from previous page

extract_attachment
get
info
remove
save

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

Appends a new data group.

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 = MDF4('new.mf4')
>>> mdf.append([s1, s2, s3], 'created by asammdf v1.1.0')
>>> # case 2: VTAB conversions from channels inside another file
>>> mdf1 = MDF4('in.mf4')
>>> ch1 = mdf1.get("Channel1_VTAB")
>>> ch2 = mdf1.get("Channel2_VTABR")
>>> sigs = [ch1, ch2]
>>> mdf2 = MDF4('out.mf4')
>>> 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 `load_measured_data=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 attachemnt *index* data. If it is an embedded attachment, then this method creates the new file according to the attachemnt 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*)

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

- using the first positional argument *name*

–if there are multiple occurances for this channel then the *group* and *index* arguments can be used to select a specific group.

–if there are multiple occurances 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

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**

info()
get MDF information as a dict

Examples

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

remove (*group=None, name=None*)

Remove data group. Use *group* or *name* keyword arguments to identify the group's index. *group* has priority

Parameters **name** : string

name of the channel inside the data group to be removed

group : int

data group index to be removed

Examples

```
>>> mdf = MDF4('test.mdf')
>>> mdf.remove(group=3)
>>> mdf.remove(name='VehicleSpeed')
```

save (*dst='', overwrite=False, 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 *'_xx'*, were *'xx'* 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; only 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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
extract	
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

Channel Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

ChannelConversion Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

ChannelGroup Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

DataGroup Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	

Continued on next page

Table 6.20 – continued from previous page

pop
popitem
setdefault
update
values

DataList Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

DataBlock Class

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

Parameters **address** : int

DTBLOCK address inside the file

file_stream : int

file handle

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	

Continued on next page

Table 6.22 – continued from previous page

setdefault
update
values

FileIdentificationBlock Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

HeaderBlock Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

SourceInformation Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

FileHistory Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
fromkeys	
get	
items	
keys	
pop	
popitem	
setdefault	
update	
values	

TextBlock Class

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

Methods

clear	
copy	Generic (shallow and deep) copying operations.
from_text	
fromkeys	
get	
items	

Continued on next page

Table 6.27 – continued from previous page

keys
pop
popitem
setdefault
update
values

classmethod `from_text` (*text*, *meta=False*)

Create a TextBlock from a str or bytes

Parameters `text` : str | bytes

input text

meta : bool

enable meta text block

Examples

```
>>> t = TextBlock.from_text(b'speed')
>>> t['id']
b'##TX'
>>> t.text_str
speed
>>> t = TextBlock.from_text('mass', meta=True)
>>> t['id']
b'##MD'
```

Notes about *load_measured_data* argument

By default when the *MDF* object is created the raw channel data is loaded into RAM. This will give you the best performance from *asammdf*.

However if you reach the physical memory limit *asammdf* gives you the option use the *load_measured_data* flag. In this case the raw channel data is not read.

MDF defaults

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 with *load_measured_data*

Advantages

- lowest RAM usage
- 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

Note: See benchmarks for the effects of using the flag

Signal

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

The Signal represents a signal described by it's 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

conversion : dict

dict describing the channel conversion , default *None*

comment : str

signal comment, default ''

Methods

astype
cut
extend
interp
plot

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

Examples

Working with MDF

```
from asammdf import MDF, Signal
import numpy as np

# create 3 Signal objects

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

# unit8
```

```

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.int32),
                  timestamps=timestamps,
                  name='Float64_Signal',
                  unit='f8')

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

# 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')

# convert new file to mdf version 3.10 with compression of raw channel data
mdf3 = mdf4.convert(to='3.10', compression=True)
print(mdf3.version)
# prints >>> 3.10

# get the float signal
sig = mdf3.get('Float64_Signal')
print(sig)
# prints >>> Signal { name="Float64_Signal":          s=[-20 -10   0  10  20] t=[ 0.1  0.2
↪      0.2          0.30000001  0.40000001  0.5          ] unit="f8"
↪ conversion=None }

```

Working with Signal

```

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')

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()
```

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 .

Intro

The benchmarks were done using two test files (for mdx version 3 and 4) of around 170MB. The files contain 183 data groups and a total of 36424 channels.

asammdf 2.5.3 was compared against *mdfreader* 0.2.5 (latest versions from PyPI). *mdfreader* seems to be the most used Python package to handle MDX 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.

Dependencies

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

- psutil
- mdfreader

x64 Python results

The test environment used for 64 bit tests had:

- 3.6.2 (v3.6.2:5fd33b5, Jul 8 2017, 04:57:36) [MSC v.1900 64 bit (AMD64)]
- Windows-10-10.0.14393-SP0

- Intel64 Family 6 Model 94 Stepping 3, GenuineIntel
- 16GB installed RAM

Notations used in the results:

- nodata = asammdf MDF object created with load_measured_data=False (raw channel data not loaded into RAM)

Files used for benchmark:

- 183 groups
- 36424 channels

Raw data

Open file	Time [ms]	RAM [MB]
asammdf 2.5.3 mdv3	876	357
asammdf 2.5.3 nodata mdv3	636	181
mdfreader 0.2.5 mdv3	3295	537
asammdf 2.5.3 mdv4	1889	436
asammdf 2.5.3 nodata mdv4	1498	245
mdfreader 0.2.5 mdv4	34732	748

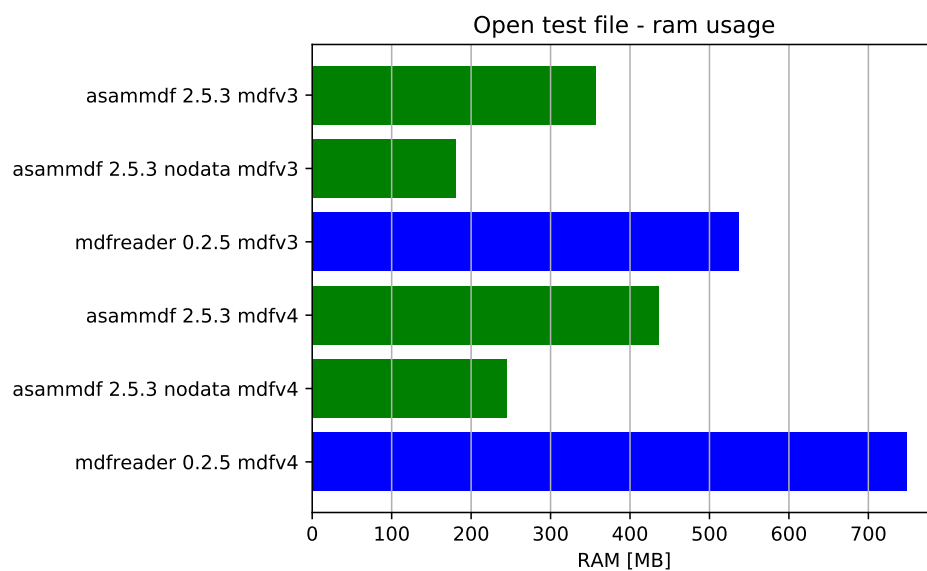
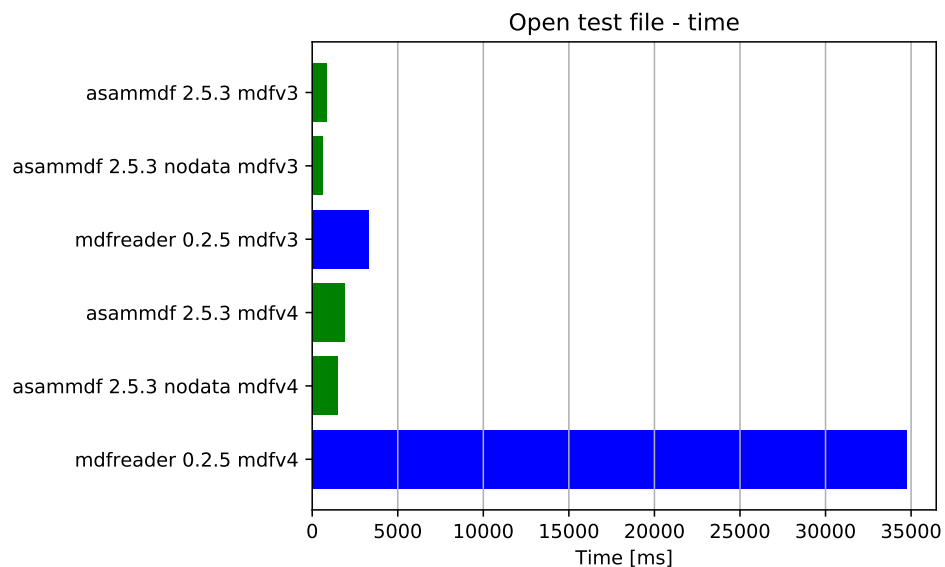
Save file	Time [ms]	RAM [MB]
asammdf 2.5.3 mdv3	486	359
asammdf 2.5.3 nodata mdv3	538	188
mdfreader 0.2.5 mdv3	25780	1996
asammdf 2.5.3 mdv4	628	442
asammdf 2.5.3 nodata mdv4	579	257
mdfreader 0.2.5 mdv4	21399	2795

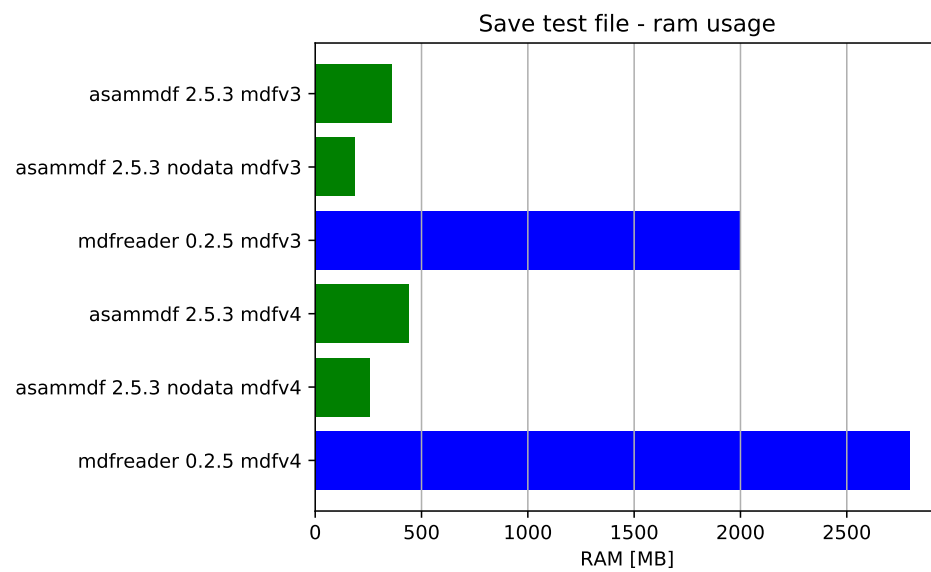
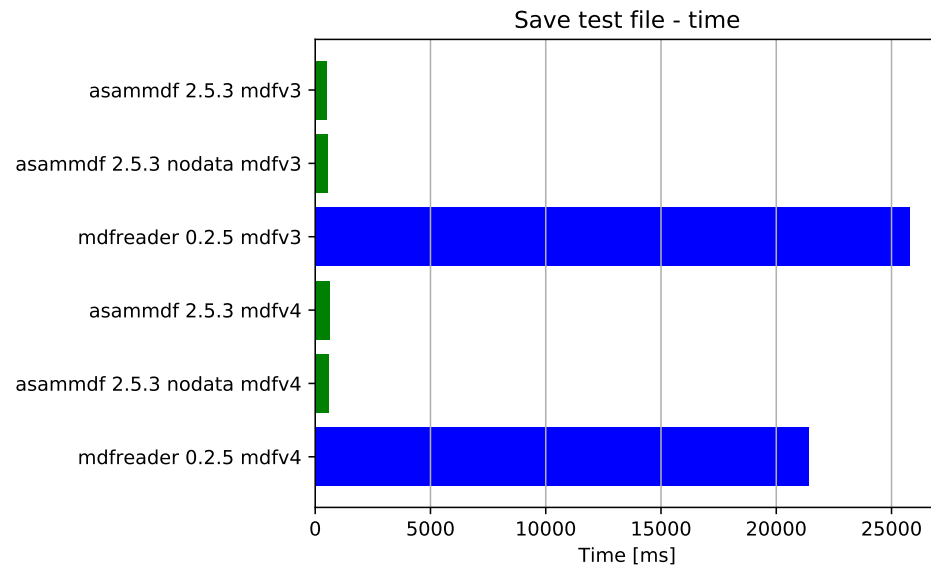
Get all channels (36424 calls)	Time [ms]	RAM [MB]
asammdf 2.5.3 mdv3	732	367
asammdf 2.5.3 nodata mdv3	10205	198
mdfreader 0.2.5 mdv3	35	537
asammdf 2.5.3 mdv4	688	445
asammdf 2.5.3 nodata mdv4	14187	258
mdfreader 0.2.5 mdv4	45	748

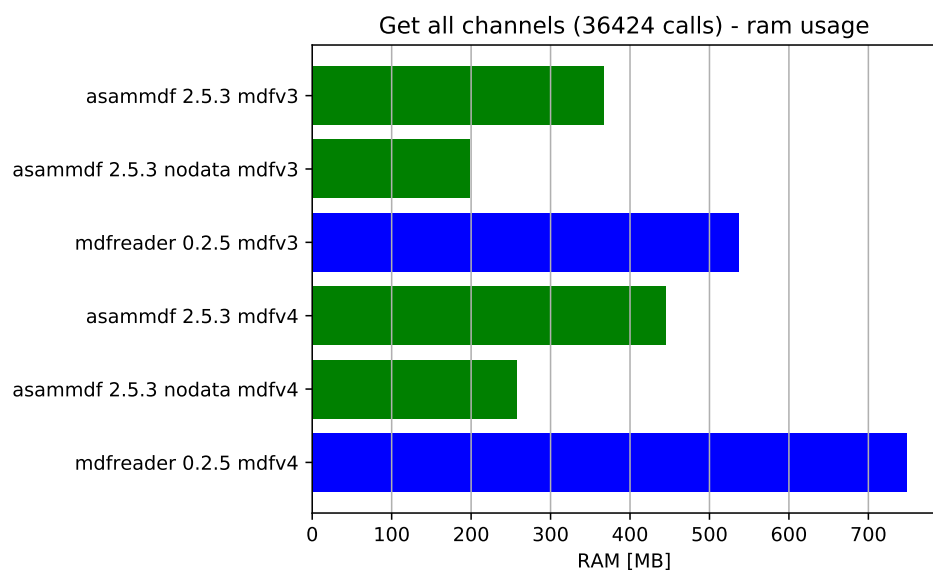
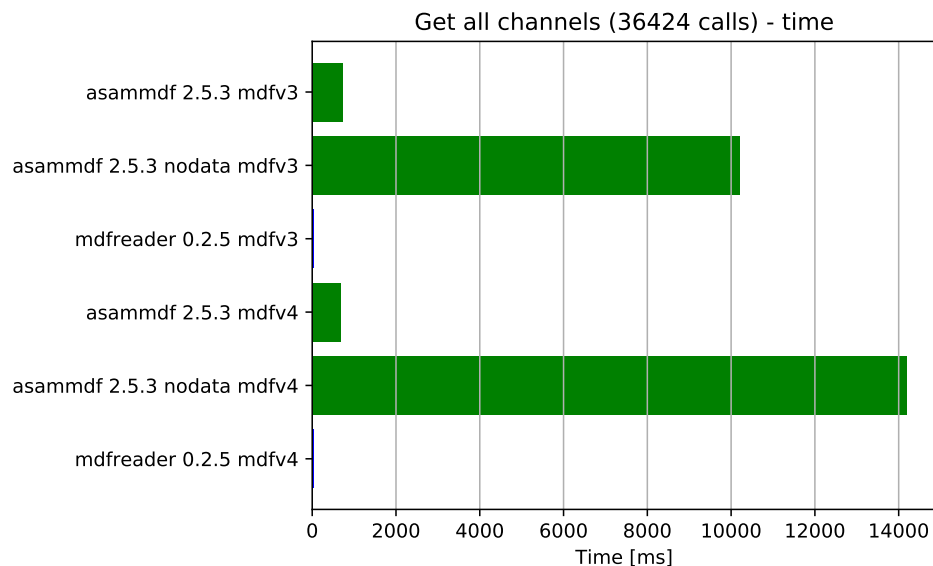
Convert file	Time [ms]	RAM [MB]
asammdf 2.5.3 v3 to v4	5056	828
asammdf 2.5.3 v3 to v4 nodata	24569	576
asammdf 2.5.3 v4 to v3	5300	851
asammdf 2.5.3 v4 to v3 nodata	29128	725

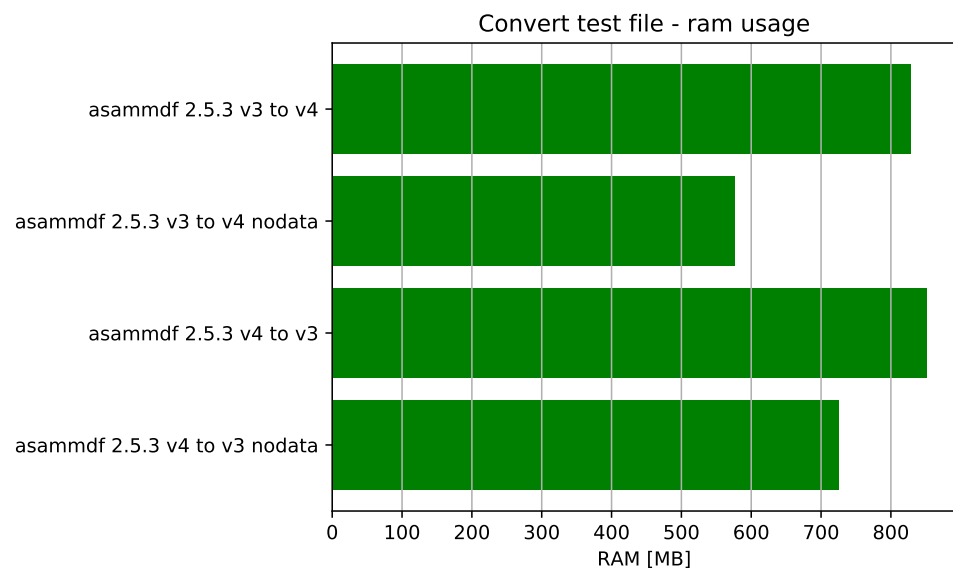
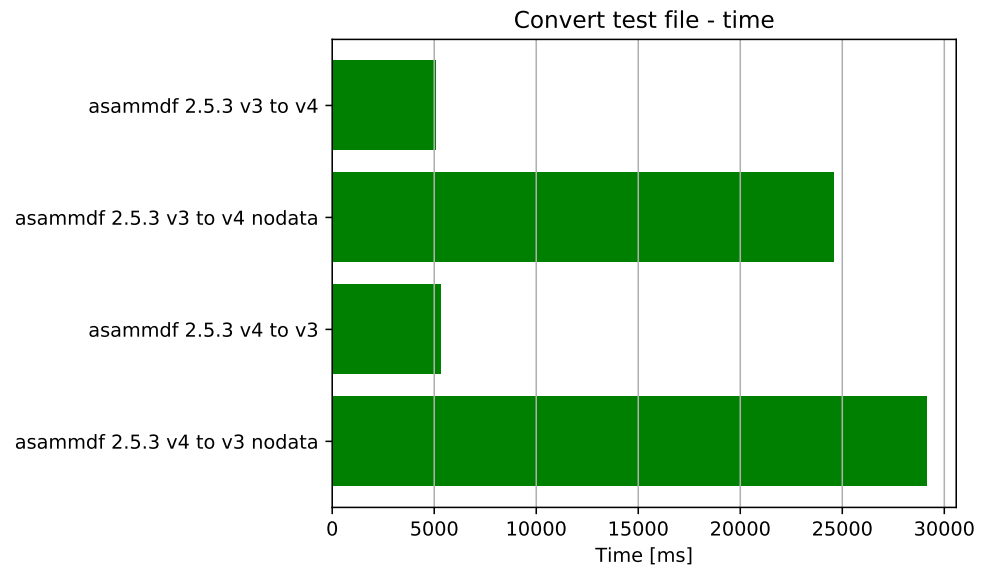
Merge files	Time [ms]	RAM [MB]
asammdf 2.5.3 v3	11408	1387
asammdf 2.5.3 v3 nodata	35575	487
asammdf 2.5.3 v4	14531	1507
asammdf 2.5.3 v4 nodata	44399	568

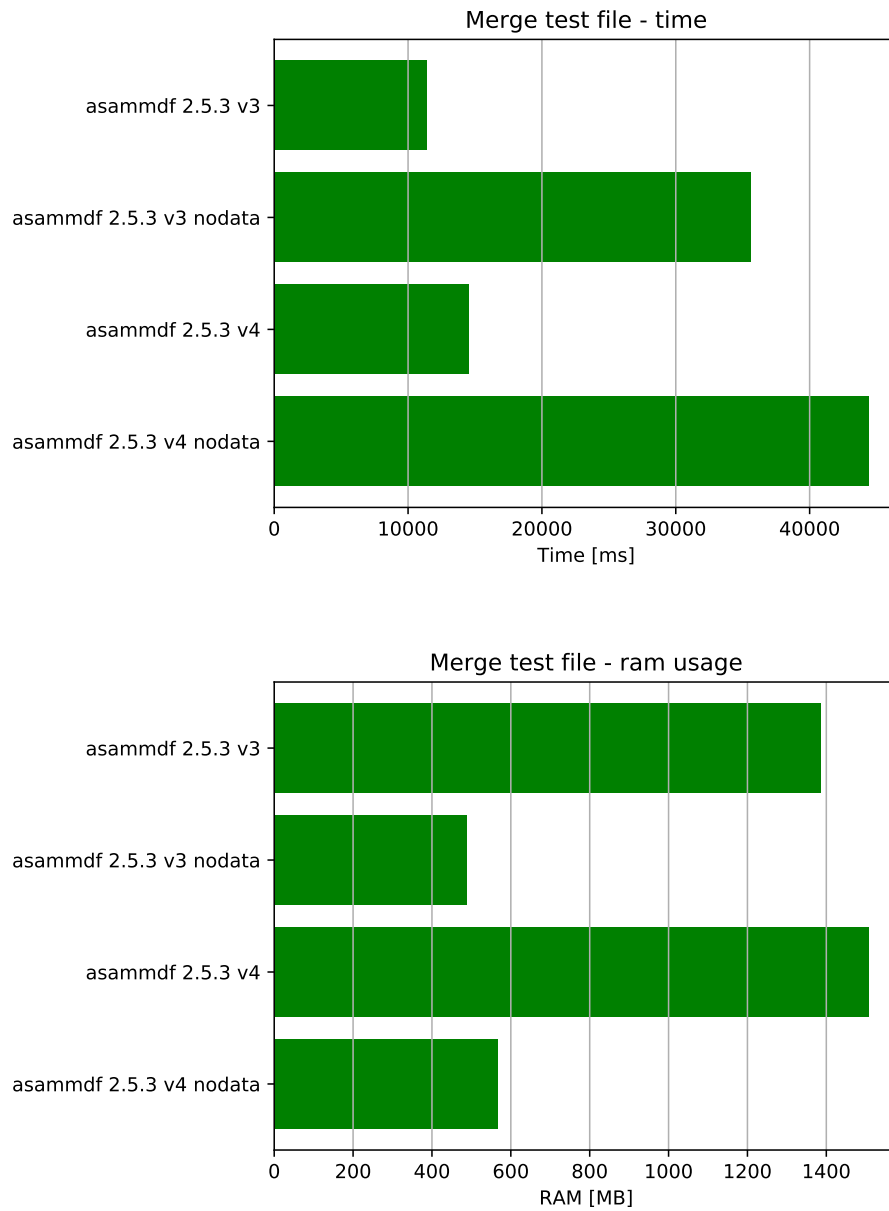
Graphical results











x86 Python results

The test environment used for 32 bit tests had:

- 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
- Windows-10-10.0.14393-SP0
- Intel64 Family 6 Model 94 Stepping 3, GenuineIntel
- 16GB installed RAM

Notations used in the results:

- nodata = asammdf MDF object created with load_measured_data=False (raw channel data not loaded into RAM)

Files used for benchmark:

- 183 groups
- 36424 channels

Raw data

Open file	Time [ms]	RAM [MB]
asammdf 2.5.3 mdfv3	897	281
asammdf 2.5.3 nodata mdfv3	648	112
mdfreader 0.2.5 mdfv3	3836	454
asammdf 2.5.3 mdfv4	2098	331
asammdf 2.5.3 nodata mdfv4	1588	151
mdfreader 0.2.5 mdfv4	45415	577

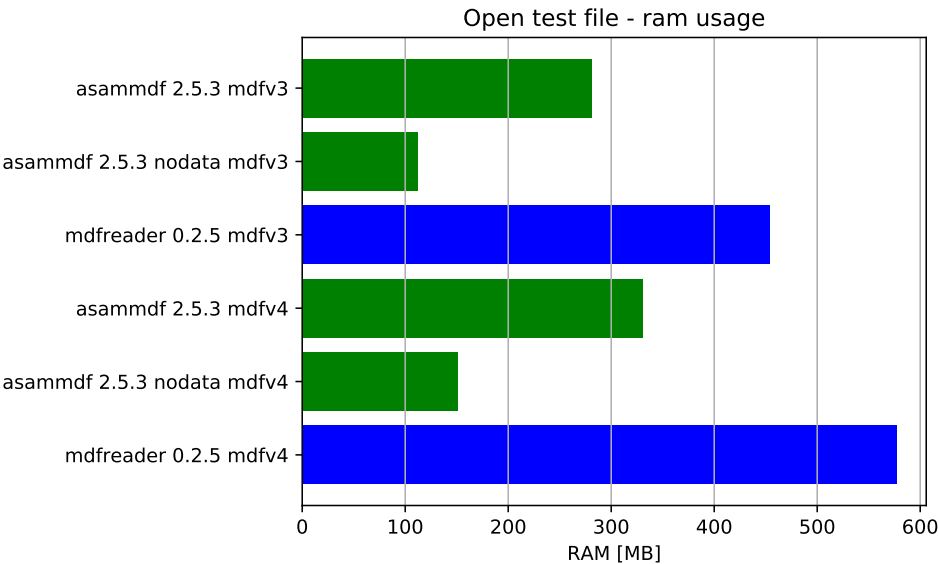
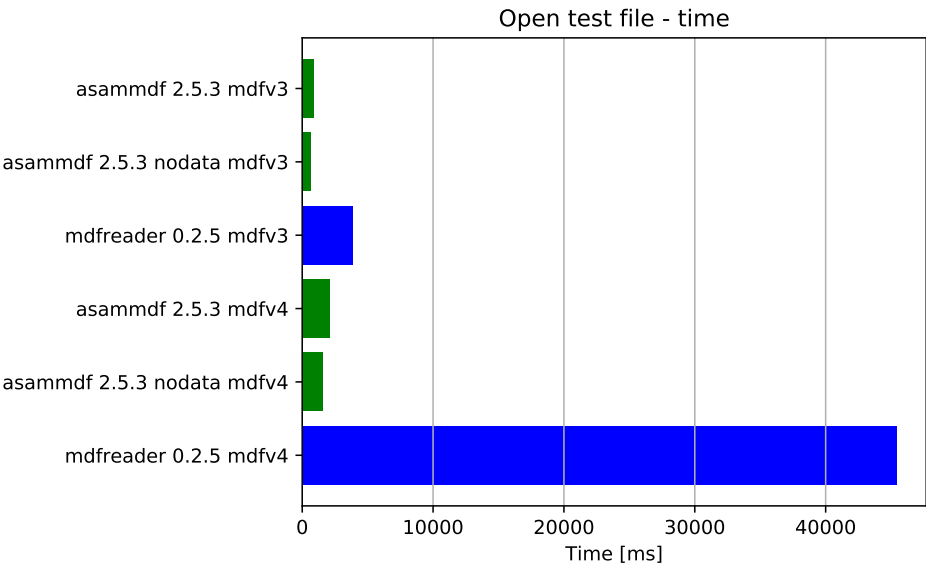
Save file	Time [ms]	RAM [MB]
asammdf 2.5.3 mdfv3	469	285
asammdf 2.5.3 nodata mdfv3	526	119
mdfreader 0.2.5 mdfv3	20328	1224
asammdf 2.5.3 mdfv4	752	337
asammdf 2.5.3 nodata mdfv4	751	160
mdfreader 0.2.5 mdfv4	18135	1686

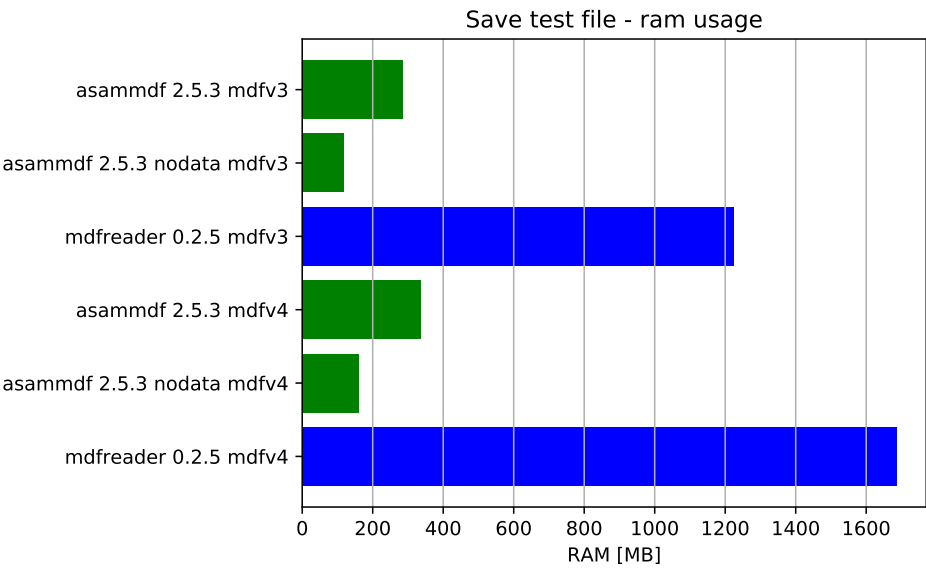
Get all channels (36424 calls)	Time [ms]	RAM [MB]
asammdf 2.5.3 mdfv3	846	289
asammdf 2.5.3 nodata mdfv3	19460	126
mdfreader 0.2.5 mdfv3	37	454
asammdf 2.5.3 mdfv4	809	337
asammdf 2.5.3 nodata mdfv4	20778	161
mdfreader 0.2.5 mdfv4	49	577

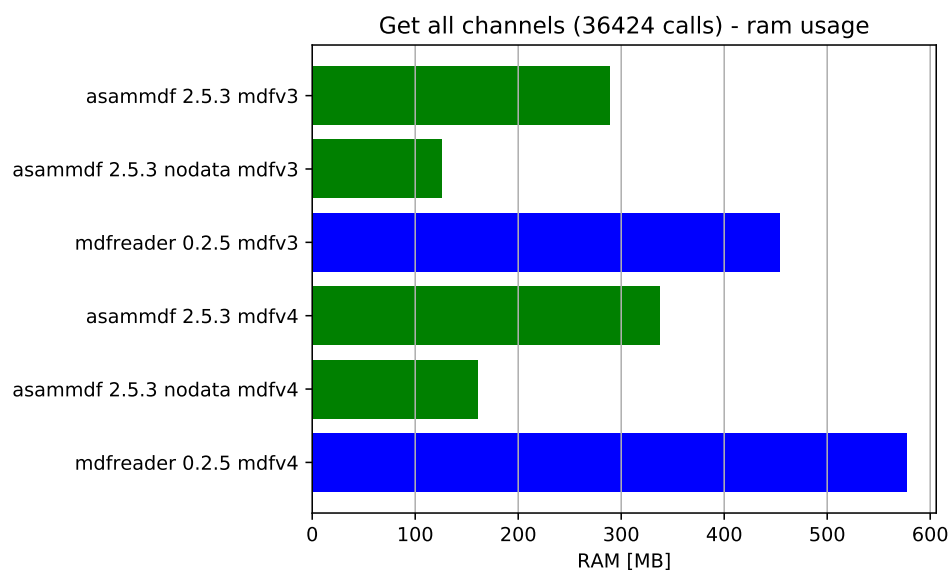
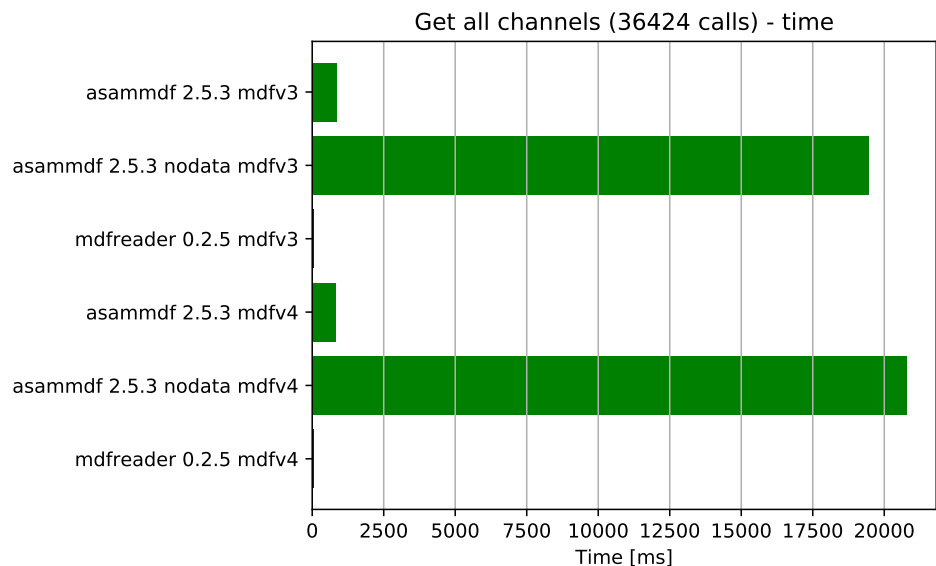
Convert file	Time [ms]	RAM [MB]
asammdf 2.5.3 v3 to v4	6121	673
asammdf 2.5.3 v3 to v4 nodata	29340	476
asammdf 2.5.3 v4 to v3	5645	690
asammdf 2.5.3 v4 to v3 nodata	32115	628

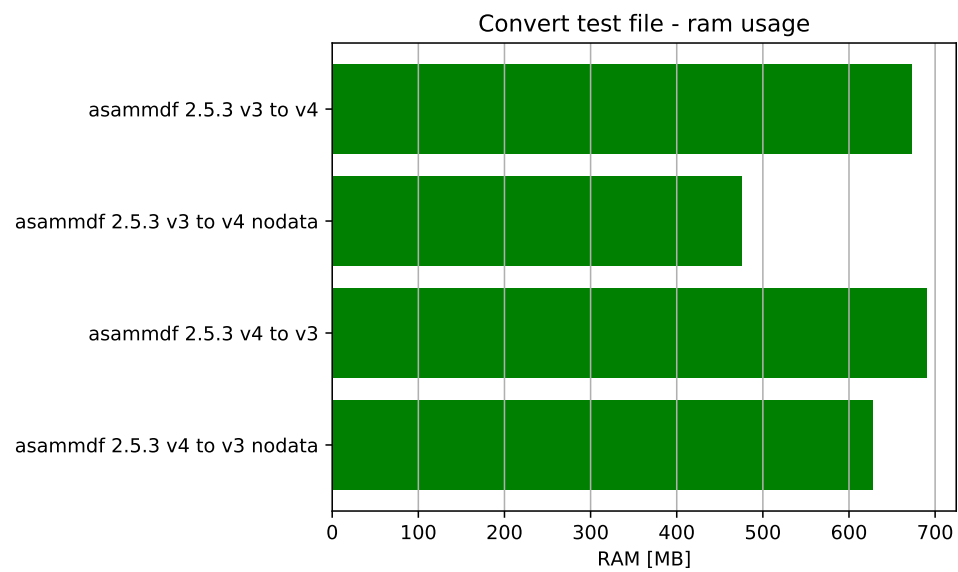
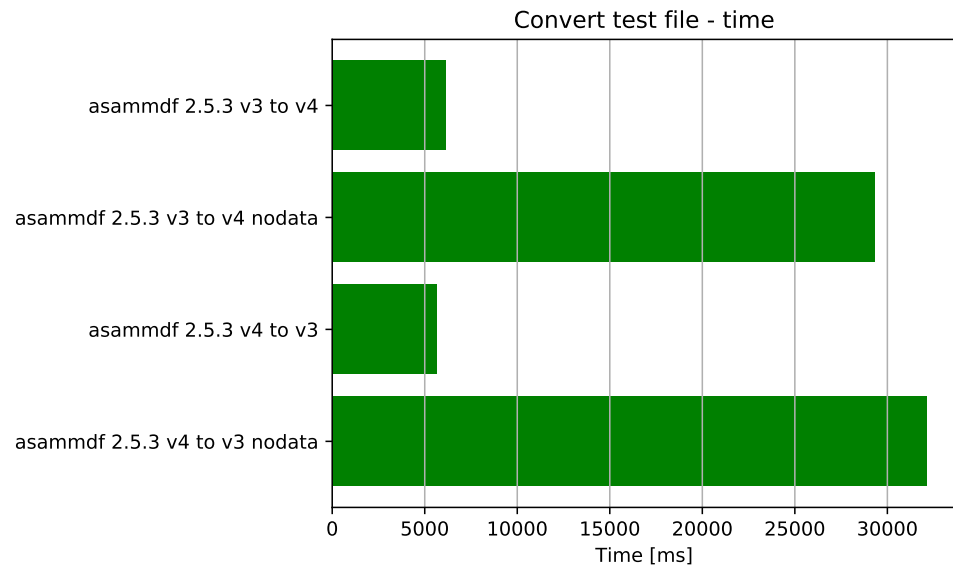
Merge files	Time [ms]	RAM [MB]
asammdf 2.5.3 v3	13392	1201
asammdf 2.5.3 v3 nodata	54040	327
asammdf 2.5.3 v4	15031	1265
asammdf 2.5.3 v4 nodata	60397	364

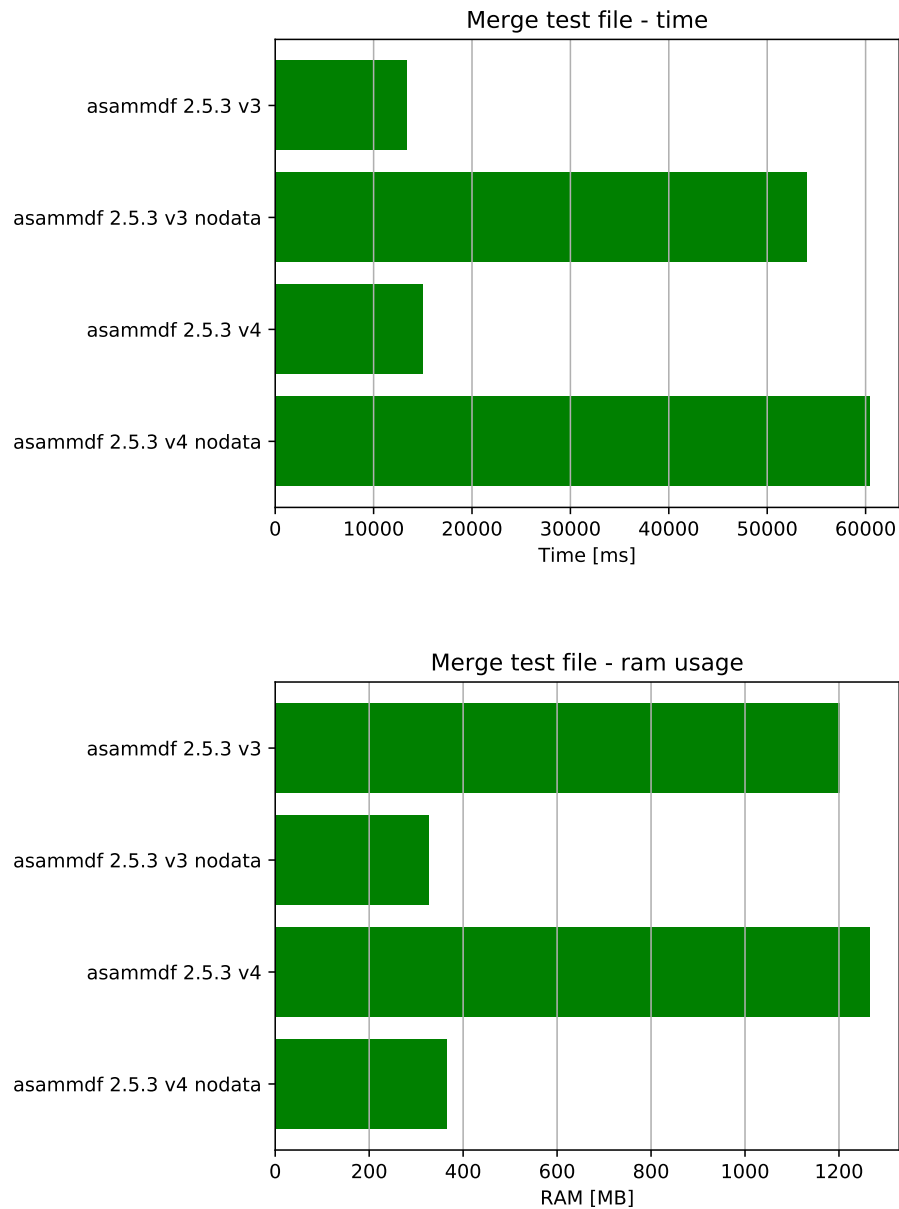
Graphical results











CHAPTER 8

Indices and tables

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

A

add_trigger() (asammdf.mdf3.MDF3 method), 17
append() (asammdf.mdf3.MDF3 method), 17
append() (asammdf.mdf4.MDF4 method), 36
astype() (asammdf.signal.Signal method), 46
attach() (asammdf.mdf4.MDF4 method), 36
AttachmentBlock (class in asammdf.mdf4), 39

C

Channel (class in asammdf.mdf3), 20
Channel (class in asammdf.mdf4), 39
ChannelConversion (class in asammdf.mdf3), 21
ChannelConversion (class in asammdf.mdf4), 39
ChannelDependency (class in asammdf.mdf3), 23
ChannelExtension (class in asammdf.mdf3), 24
ChannelGroup (class in asammdf.mdf3), 25
ChannelGroup (class in asammdf.mdf4), 40
close() (asammdf.mdf3.MDF3 method), 18
close() (asammdf.mdf4.MDF4 method), 37
convert() (asammdf.mdf.MDF method), 14
cut() (asammdf.mdf.MDF method), 14
cut() (asammdf.signal.Signal method), 46

D

DataBlock (class in asammdf.mdf4), 41
DataGroup (class in asammdf.mdf3), 27
DataGroup (class in asammdf.mdf4), 40
DataList (class in asammdf.mdf4), 41

E

export() (asammdf.mdf.MDF method), 14
extend() (asammdf.signal.Signal method), 46
extract_attachment() (asammdf.mdf4.MDF4 method), 37

F

FileHistory (class in asammdf.mdf4), 43
FileIdentificationBlock (class in asammdf.mdf3), 28
FileIdentificationBlock (class in asammdf.mdf4), 42
filter() (asammdf.mdf.MDF method), 14

from_text() (asammdf.mdf4.TextBlock class method), 44

G

get() (asammdf.mdf3.MDF3 method), 18
get() (asammdf.mdf4.MDF4 method), 37

H

HeaderBlock (class in asammdf.mdf3), 29
HeaderBlock (class in asammdf.mdf4), 42

I

info() (asammdf.mdf3.MDF3 method), 18
info() (asammdf.mdf4.MDF4 method), 38
interp() (asammdf.signal.Signal method), 46
iter_get_triggers() (asammdf.mdf3.MDF3 method), 19
iter_to_pandas() (asammdf.mdf.MDF method), 15

M

MDF (class in asammdf.mdf), 13
MDF3 (class in asammdf.mdf3), 16
MDF4 (class in asammdf.mdf4), 35
merge() (asammdf.mdf.MDF static method), 15

P

plot() (asammdf.signal.Signal method), 46
ProgramBlock (class in asammdf.mdf3), 30

R

remove() (asammdf.mdf3.MDF3 method), 19
remove() (asammdf.mdf4.MDF4 method), 38

S

SampleReduction (class in asammdf.mdf3), 31
save() (asammdf.mdf3.MDF3 method), 19
save() (asammdf.mdf4.MDF4 method), 38
Signal (class in asammdf.signal), 45
SourceInformation (class in asammdf.mdf4), 42

T

[TextBlock \(class in asammdf.mdf3\)](#), [32](#)

[TextBlock \(class in asammdf.mdf4\)](#), [43](#)

[TriggerBlock \(class in asammdf.mdf3\)](#), [33](#)