asammdf Documentation

Release 2.0.0pre

Daniel Hrisca

Jul 31, 2017

Contents

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

asammdf supports mdf versions 3 and 4 and can be used with 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
- clean and simple data types

CHAPTER 2

Features

- read sorted and unsorted MDF v3 and v4 files
- files are loaded in RAM for fast operations
  - for low memory computers or for large data files there is the option to load only the metadata and leave the raw channel data (the samples) unread; this of course will mean slower channel data access speed
- 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 measuremetn will have channels from different sources at different rates
  - the *Signal* class facilitates operations with such channels
- remove data group by index or by specifing a channel name inside the target data group
- append new channels
- convert to different mdf version

CHAPTER 3

Major features still not implemented

- functionality related to sample reduction block (but the class is defined)
- mdf 3 channel dependency functionality
- functionality related to trigger blocks (but the class is defined)
- handling of unfinnished measurements (mdf 4)
- compressed data blocks for mdf >= 4.10
- mdf 4 attachment blocks
- mdf 4 channel arrays
- mdf 4 VLSD channels and SDBLOCKs
- xml schema for TXBLOCK and MDBLOCK

CHAPTER 4

Dependencies

asammdf uses the following libraries

- numpy : the heart that makes all tick
- numexpr : for formula based channel conversions
- blosc : optionally used for in memmory raw channel data compression
- matplotlib : for Signal plotting

CHAPTER 5

Features

- read sorted and unsorted MDF v3 files
- **files are loaded in RAM for fast operations**
  - for low memory computers or for large data files there is the option to load only the metadata and leave the raw channel data (the samples) unread; this of course will mean slower channel data access speed
- extract channel data, master channel and extra channel information (unit, conversion rule)
- remove data group by index or by specifing a channel name inside the target data group
- append new channels
- convert to different mdf version

CHAPTER 6

Major features still not implemented

- functionality related to sample reduction block (but the class is defined)
- mdf 3 channel dependency functionality
- functionality related to trigger blocks (but the class is defined)
- handling of unfinnished measurements (mdf 4)
- compressed data blocks for mdf >= 4.10
- mdf 4 attachment blocks
- mdf 4 channel arrays
- mdf 4 VLSD channels and SDBLOCKs
- xml schema for TXBLOCK and MDBLOCK

CHAPTER 7

Installation

asammdf is available on

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

```
pip install asammdf
```

CHAPTER 8

API

MDF

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

MDF3 and MDF4 classes

MDF3

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

The *header* attibute 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 : dictionay containing TextBlock objects used throughout the mdf
  - channels : list of dictionaries that contain TextBlock objects ralated 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 ralated to each channel group
    * comment_addr : channel group comment
  - conversion_tab : list of dictionaries that contain TextBlock objects ralated to VATB and VTABR channel conversions
    * text_{n} : n-th text of the VTABR conversion

The *file_history* attribute is a TextBlock object.

The *channel_db* attibute 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* attibute 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

MDF version 3 blocks

The following classes implement different MDF version3 blocks.

Channel Class

ChannelConversion Class

ChannelDependency Class

ChannelExtension Class

ChannelGroup Class

DataGroup Class

FileIdentificationBlock Class

HeaderBlock Class

ProgramBlock Class

SampleReduction Class

TextBlock Class

TriggerBlock Class

MDF4

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

The *header* attibute 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 : dictionay containing TextBlock objects used throughout the mdf
  - channels : list of dictionaries that contain TextBlock objects ralated to each channel
    - name_addr : channel name
    - comment_addr : channel comment
  - channel group : list of dictionaries that contain TextBlock objects ralated 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 containt TextBlock obejcts related to channel conversions
    - name_addr : converions name
    - unit_addr : channel unit_addr
    - comment_addr : converison comment
    - formula_addr : formula text; only valid for algebraic conversions
  - sources : list of dictionaries that containt TextBlock obejcts 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* attibute 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* attibute 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

MDF version 4 blocks

The following classes implement different MDF version3 blocks.

Channel Class

ChannelConversion Class

ChannelGroup Class

DataGroup Class

DataList Class

DataBlock Class

FileIdentificationBlock Class

HeaderBlock Class

SourceInformation Class

FileHistory Class

TextBlock Class

Notes about *compression* and *load_measured_data* arguments

By default *MDF* object use no compression and the raw channel data is loaded into RAM. This will give you the best performance from *asammdf*.

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

1. use the *compression* flag: raw channel data is loaded into RAM but it is compressed. The default compression library is *blosc* and as a fallback *zlib* is used (slower). The advange is that you save RAM, but in return you will pay the compression/decompression time penalty in all operations (file open, getting channel data, saving to disk, converting).
2. use the *load_measured_data* flag: raw channel data is not read.

MDF defaults

Advantages

- best performance

Disadvantages

- highest RAM usage

Use case

- when data fits inside the system RAM

MDF with *compression*

Advantages

- lower RAM usage than *default*
- alows saving to disk and appending new data

Disadvantages

- slowest

Use case

- when *default* data exceeds RAM and you need to append and save

MDF with *load_measured_data*

Advantages

- lowest RAM usage
- faster than *compression*

Disadvantages

- ReadOnly mode: appending and saving is not possible

Use case

- when *default* data exceeds RAM and you only want to extract information from the file

---

Note: See benchmarks for the effects of using the flags.

Signal

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         0.30000001  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

# unit8 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()
```

CHAPTER 9

Benchmarks

Intro

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

asamdf 2.0.0 was compared against *mdfreader 0.2.5*. *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 cathegories are file open, file save and extracting the data for all channels inside the file(36424 calls). For each cathegory two aspect 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:

- Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 18:41:36) [MSC v.1900 64 bit (AMD64)]
- Windows-7-6.1.7601-SP1
- Intel64 Family 6 Model 94 Stepping 3, GenuineIntel (i7-6820Q)
- 16GB installed RAM

The notations used in the results have the following meaning:

- nodata = MDF object created with load_measured_data=False (raw channel data no loaded into RAM)
- compression = MDF object created with compression=True (raw channel data loaded into RAM and compressed)
- noconvert = MDF object created with convertAfterRead=False

Raw data

Open file	Time [ms]	RAM [MB]
asammdf 2.0.0 mdfv3	721	352
asammdf 2.0.0 compression mdfv3	1008	275
asammdf 2.0.0 nodata mdfv3	641	199
mdfreader 0.2.5 mdfv3	2996	526
mdfreader 0.2.5 no convert mdfv3	2846	393
asammdf 2.0.0 mdfv4	1634	439
asammdf 2.0.0 compression mdfv4	1917	343
asammdf 2.0.0 nodata mdfv4	1594	274
mdfreader 0.2.5 mdfv4	31023	739
mdfreader 0.2.5 noconvert mdfv4	30693	609

Save file	Time [ms]	RAM [MB]
asammdf 2.0.0 mdfv3	472	353
asammdf 2.0.0 compression mdfv3	667	275
mdfreader 0.2.5 mdfv3	18910	2003
asammdf 2.0.0 mdfv4	686	447
asammdf 2.0.0 compression mdfv4	836	352
mdfreader 0.2.5 mdfv4	16631	2802

Get all channels	Time [ms]	RAM [MB]
asammdf 2.0.0 mdfv3	2492	362
asammdf 2.0.0 compression mdfv3	14474	285
asammdf 2.0.0 nodata mdfv3	9621	215
mdfreader 0.2.5 mdfv3	31	526
asammdf 2.0.0 mdfv4	2066	450
asammdf 2.0.0 compression mdfv4	16944	359
asammdf 2.0.0 nodata mdfv4	12364	292
mdfreader 0.2.5 mdfv4	39	739

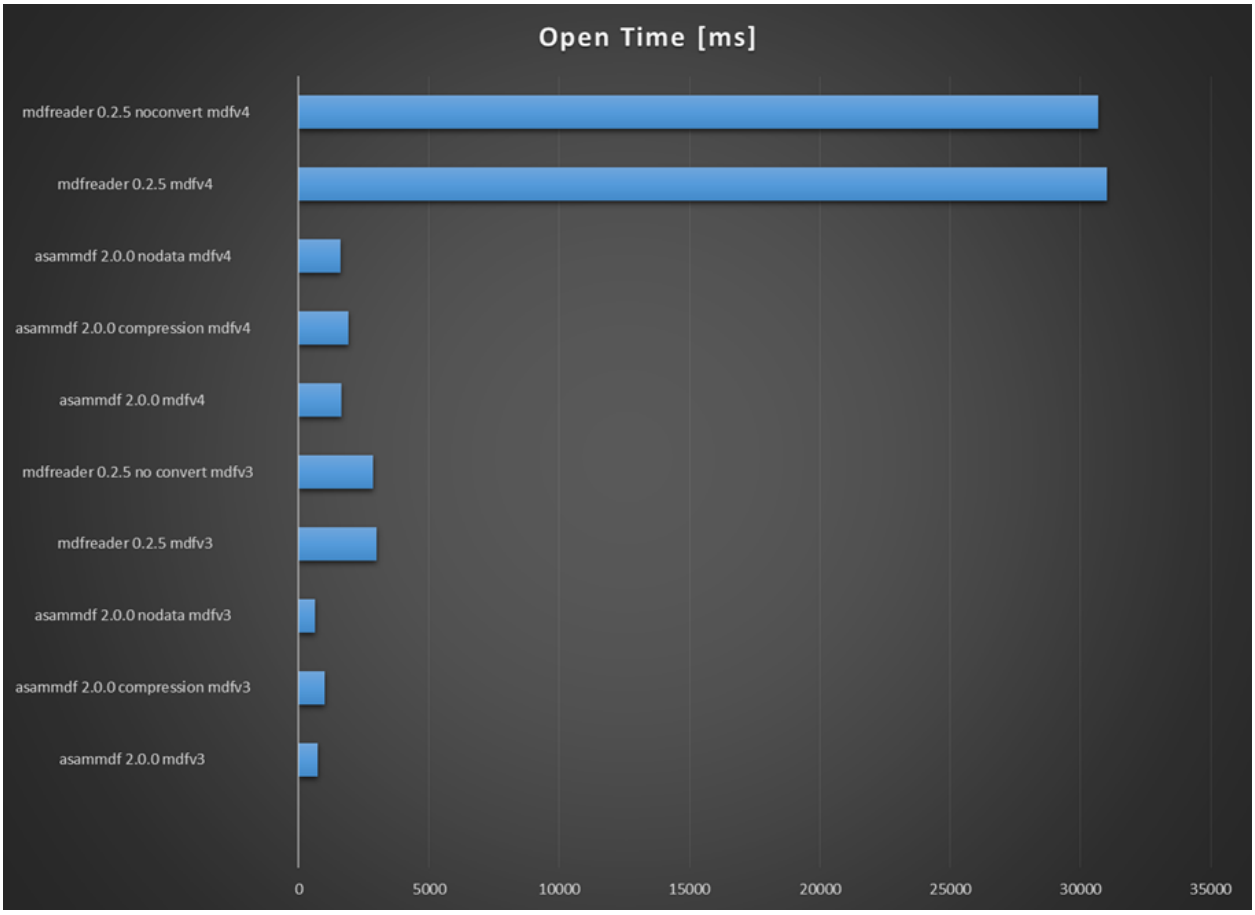
Graphical results

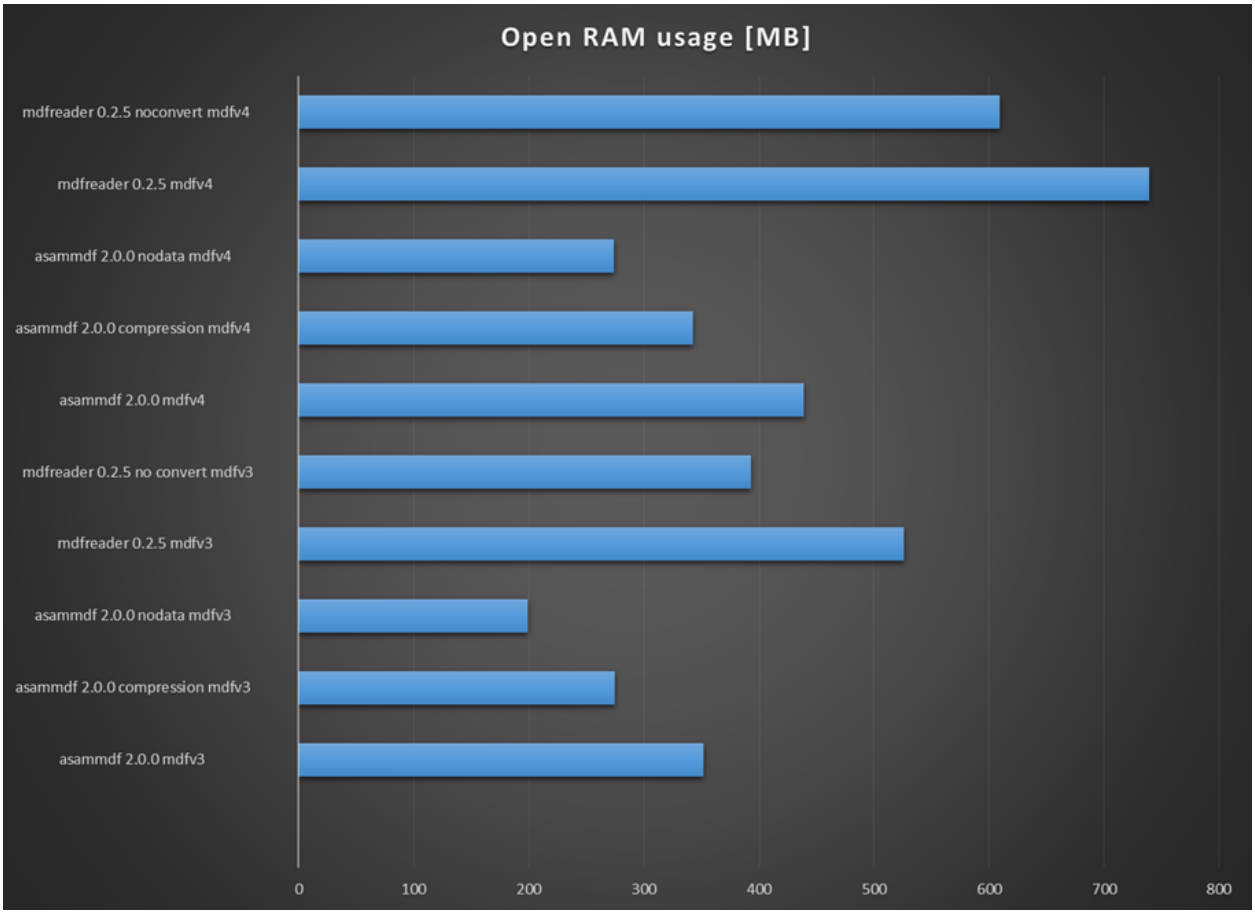
x86 Python results

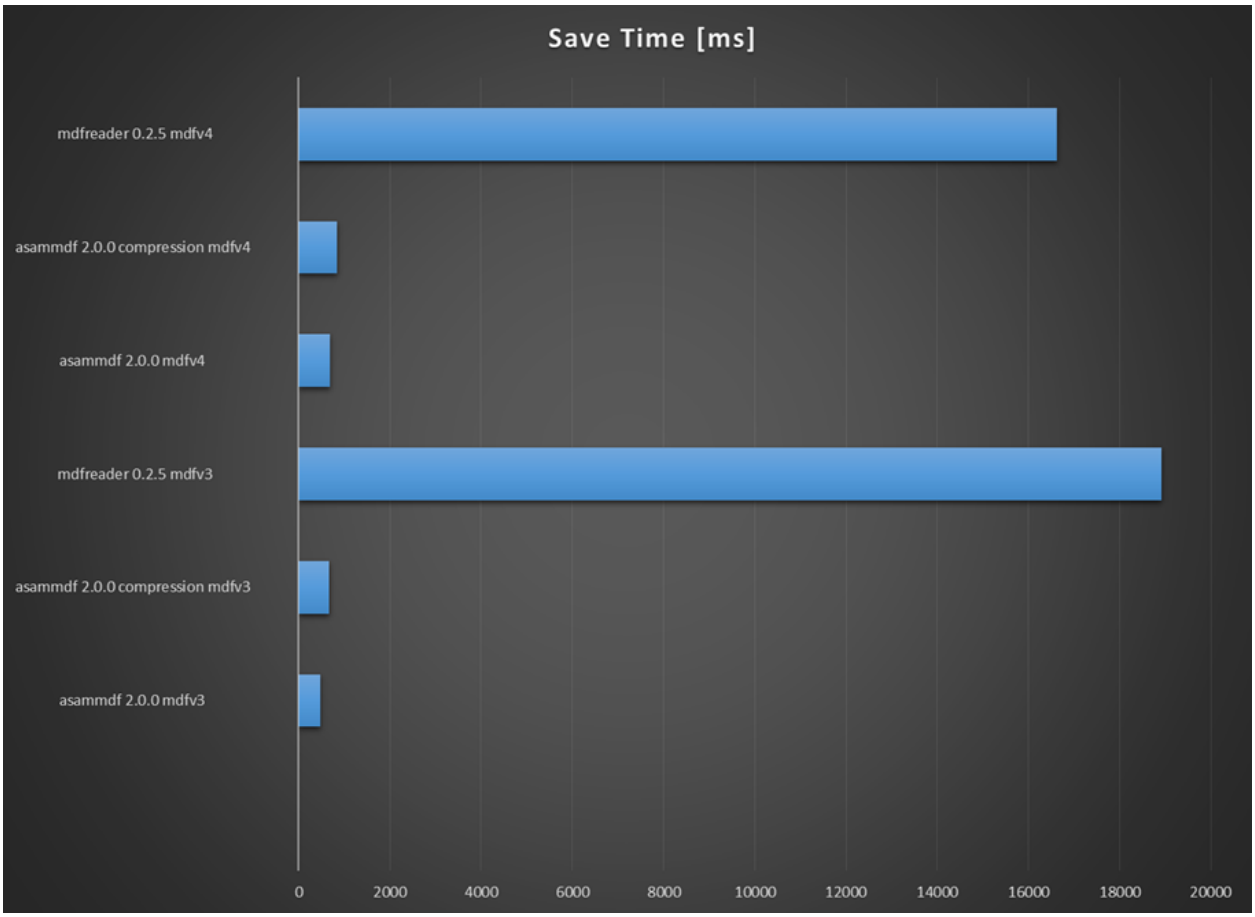
The test environment used for 32 bit tests had:

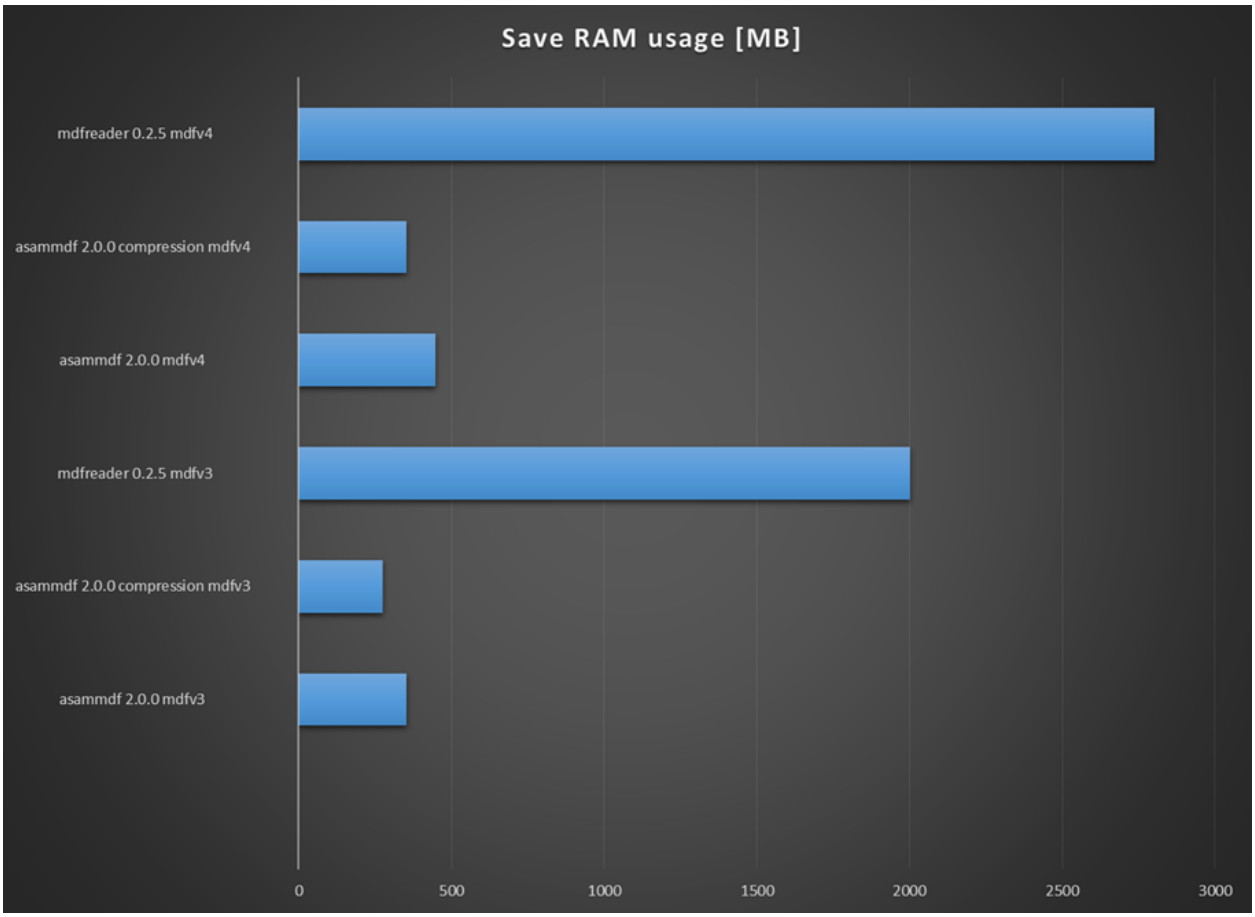
- Python 3.6.1 (v3.6.1:69c0db5, Mar 21 2017, 17:54:52) [MSC v.1900 32 bit (Intel)]
- Windows-7-6.1.7601-SP1
- Intel64 Family 6 Model 94 Stepping 3, GenuineIntel (i7-6820Q)
- 16GB installed RAM

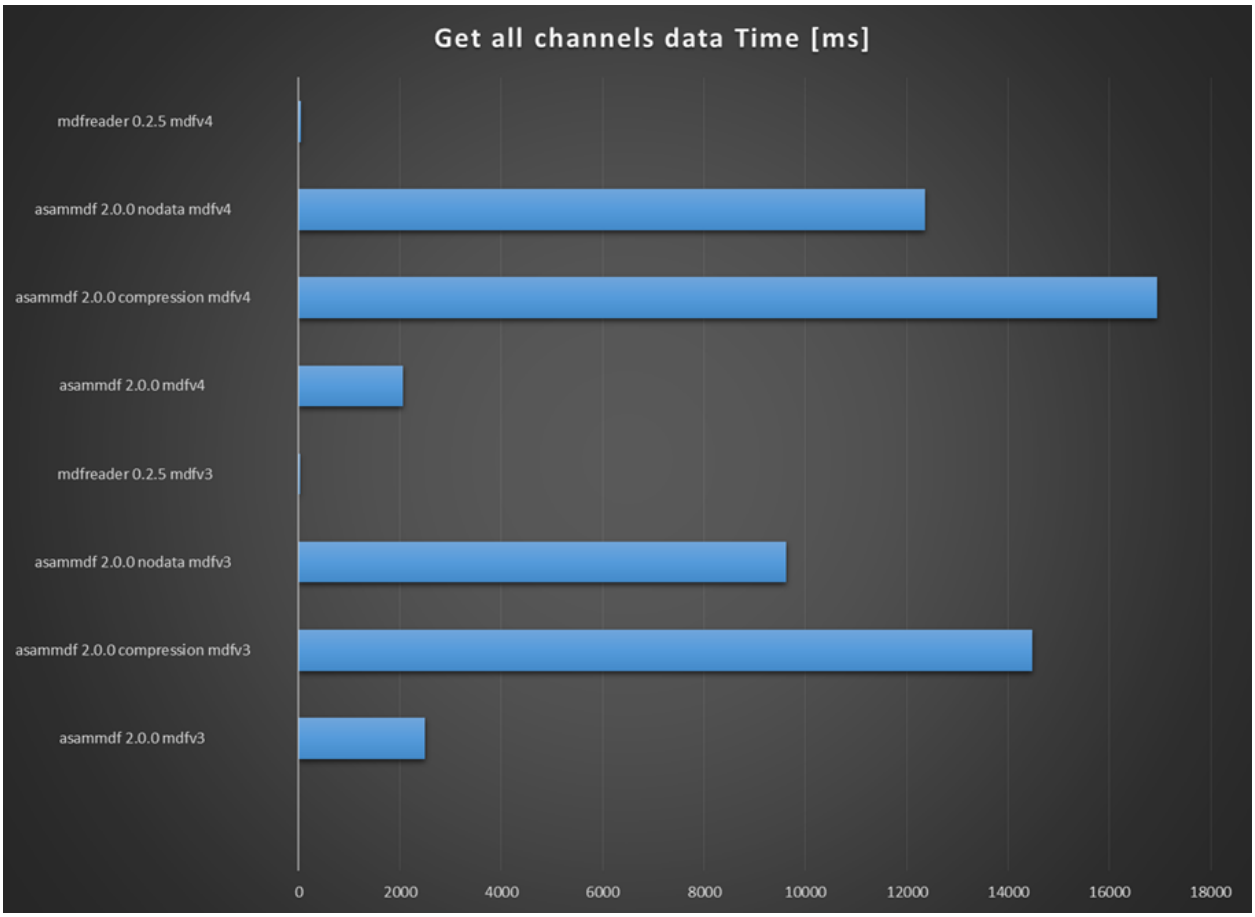
The notations used in the results have the following meaning:

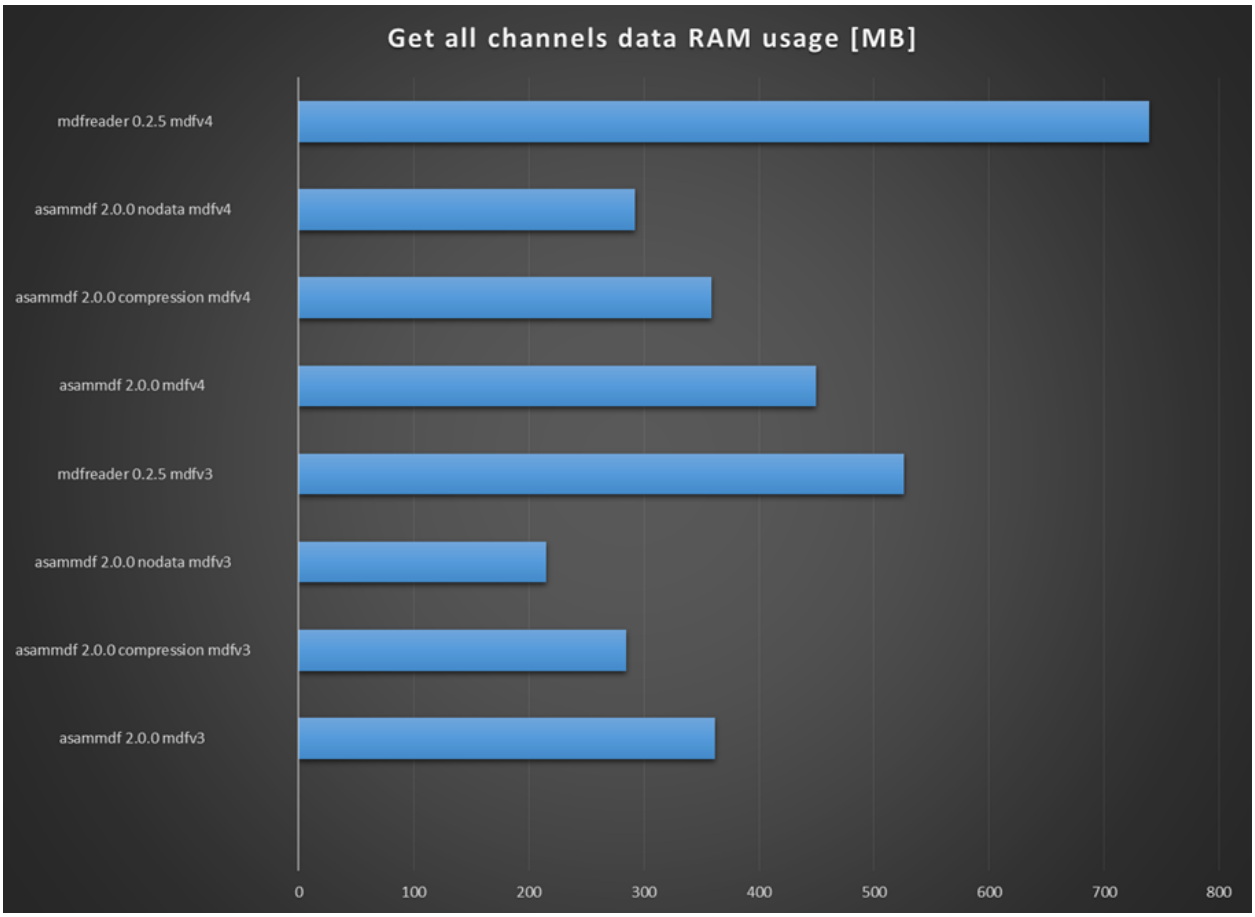

Open Time [ms]
mdfreader 0.2.5 noconvert mdfv4
mdfreader 0.2.5 mdfv4
asammdf 2.0.0 nodata mdfv4
asammdf 2.0.0 compression mdfv4
asammdf 2.0.0 mdfv4
mdfreader 0.2.5 no convert mdfv3
mdfreader 0.2.5 mdfv3
asammdf 2.0.0 nodata mdfv3
asammdf 2.0.0 compression mdfv3
asammdf 2.0.0 mdfv3
0
5000
10000
15000
20000
25000
30000
35000


Open RAM usage [MB]
mdfreader 0.2.5 noconvert mdfv4
mdfreader 0.2.5 mdfv4
asammdf 2.0.0 nodata mdfv4
asammdf 2.0.0 compression mdfv4
asammdf 2.0.0 mdfv4
mdfreader 0.2.5 no convert mdfv3
mdfreader 0.2.5 mdfv3
asammdf 2.0.0 nodata mdfv3
asammdf 2.0.0 compression mdfv3
asammdf 2.0.0 mdfv3
0
100
200
300
400
500
600
700
800


Save Time [ms]
mdfreader 0.2.5 mdfv4
asammdf 2.0.0 compression mdfv4
asammdf 2.0.0 mdfv4
mdfreader 0.2.5 mdfv3
asammdf 2.0.0 compression mdfv3
asammdf 2.0.0 mdfv3
0
2000
4000
6000
8000
10000
12000
14000
16000
18000
20000


Save RAM usage [MB]
mdfreader 0.2.5 mdfv4
asammdf 2.0.0 compression mdfv4
asammdf 2.0.0 mdfv4
mdfreader 0.2.5 mdfv3
asammdf 2.0.0 compression mdfv3
asammdf 2.0.0 mdfv3
0
500
1000
1500
2000
2500
3000


Get all channels data Time [ms]
mdfreader 0.2.5 mdfv4
asammdf 2.0.0 nodata mdfv4
asammdf 2.0.0 compression mdfv4
asammdf 2.0.0 mdfv4
mdfreader 0.2.5 mdfv3
asammdf 2.0.0 nodata mdfv3
asammdf 2.0.0 compression mdfv3
asammdf 2.0.0 mdfv3
0
2000
4000
6000
8000
10000
12000
14000
16000
18000


Get all channels data RAM usage [MB]
mdfreader 0.2.5 mdfv4
asammdf 2.0.0 nodata mdfv4
asammdf 2.0.0 compression mdfv4
asammdf 2.0.0 mdfv4
mdfreader 0.2.5 mdfv3
asammdf 2.0.0 nodata mdfv3
asammdf 2.0.0 compression mdfv3
asammdf 2.0.0 mdfv3
0
100
200
300
400
500
600
700
800

- nodata = MDF object created with load_measured_data=False (raw channel data no loaded into RAM)
- compression = MDF object created with compression=True (raw channel data loaded into RAM and compressed)
- noconvert = MDF object created with convertAfterRead=False

Raw data

Open file	Time [ms]	RAM [MB]
asammdf 2.0.0 mdfv3	851	283
asammdf 2.0.0 compression mdfv3	1149	190
asammdf 2.0.0 nodata mdfv3	765	129
mdfreader 0.2.5 mdfv3	3633	453
mdfreader 0.2.5 no convert mdfv3	3309	319
asammdf 2.0.0 mdfv4	1854	339
asammdf 2.0.0 compression mdfv4	2191	236
asammdf 2.0.0 nodata mdfv4	1772	173
mdfreader 0.2.5 mdfv4	42177	576
mdfreader 0.2.5 noconvert mdfv4	41799	447

Save file	Time [ms]	RAM [MB]
asammdf 2.0.0 mdfv3	564	286
asammdf 2.0.0 compression mdfv3	756	194
mdfreader 0.2.5 mdfv3	17499	1236
asammdf 2.0.0 mdfv4	906	347
asammdf 2.0.0 compression mdfv4	1112	244
mdfreader 0.2.5 mdfv4	15027	1698

Get all channels	Time [ms]	RAM [MB]
asammdf 2.0.0 mdfv3	3224	293
asammdf 2.0.0 compression mdfv3	25019	201
asammdf 2.0.0 nodata mdfv3	18824	144
mdfreader 0.2.5 mdfv3	35	454
asammdf 2.0.0 mdfv4	2513	349
asammdf 2.0.0 compression mdfv4	25140	250
asammdf 2.0.0 nodata mdfv4	19862	188
mdfreader 0.2.5 mdfv4	50	576

CHAPTER 10

Indices and tables

- genindex
- modindex
- search