asammdf Documentation

Release 4.0.0

Daniel Hrisca

Sep 11, 2018

Contents

1 Introduction **3**
- 1.1 Project goals . . . 3
- 1.2 Features . . . 3
- 1.3 Major features not implemented (yet) . . . 4
- 1.4 Dependencies . . . 4
- 1.5 Installation . . . 5
- 1.6 Contributing . . . 5
  - 1.6.1 Contributors . . . 5

2 API **7**
- 2.1 MDF . . . 7
- 2.2 MDF3 . . . 13
  - 2.2.1 MDF version 2 & 3 blocks . . . 20
- 2.3 MDF4 . . . 29
  - 2.3.1 MDF version 4 blocks . . . 37
- 2.4 Signal . . . 47

3 Bus logging **49**

4 Tips **51**
- 4.1 Impact of *memory* argument . . . 51
  - 4.1.1 *MDF* created with *memory='full'* . . . 51
  - 4.1.2 *MDF* created with *memory='low'* . . . 51
  - 4.1.3 *MDF* created with *memory='minimum'* . . . 52
- 4.2 Chunked data access . . . 52
- 4.3 Optimized methods . . . 53
- 4.4 Faster file loading . . . 53
  - 4.4.1 BytesIO and *memory='full'* . . . 53
  - 4.4.2 Skip XML parsing for MDF4 files . . . 53

5 Examples **55**
- 5.1 Working with MDF . . . 55
- 5.2 Working with Signal . . . 56
- 5.3 MF4 demo file generator . . . 57

6 Benchmarks **63**
- 6.1 Test setup . . . 63

6.1.1 Dependencies . . . 63
6.1.2 Usage . . . 63
6.2 x64 Python results . . . 64
6.2.1 Graphical results . . . 66
6.3 x86 Python results . . . 66
6.3.1 Graphical results . . . 70

7 GUI **75**
7.1 Menu . . . 75
7.1.1 File . . . 75
7.1.2 Settings . . . 75
7.1.3 Plot . . . 76
7.2 Single files . . . 77
7.2.1 Opened files tabs . . . 78
7.2.2 Quick channel search field for the current file . . . 78
7.2.3 Complete channels tree . . . 78
7.2.4 Command buttons . . . 79
7.2.5 Selected channels list . . . 79
7.2.6 Plot . . . 81
7.3 Multiple files . . . 88

8 Indices and tables **91**

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 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

Introduction

1.1 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

1.2 Features

- create new mdf files from scratch
- append new channels
- read unsorted MDF v3 and v4 files
- read CAN bus logging files
- filter a subset of channels from original mdf file
- cut measurement to specified time interval
- convert to different mdf version
- export to pandas, Excel, HDF5, Matlab (v4, v5 and v7.3), CSV and parquet
- merge multiple files sharing the same internal structure
- read and save mdf version 4.10 files containing zipped data blocks
- space optimizations for saved files (no duplicated blocks)
- split large data blocks (configurable size) for mdf version 4
- 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
- handle large files (for example merging two fileas, each with 14000 channels and 5GB size, on a RaspberryPi) 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
  - a measurement will usually have channels from different sources at different rates
  - the *Signal* class facilitates operations with such channels

1.3 Major features not implemented (yet)

- for version 3
  - functionality related to sample reduction block: the samples reduction blocks are simply ignored
- for version 4
  - functionality related to sample reduction block: the samples reduction blocks are simply ignored
  - handling of channel hierarchy: channel hierarchy is ignored
  - full handling of bus logging measurements: currently only CAN bus logging is implemented with the ability to *get* signals defined in the attached CAN database (.arxml or .dbc)
  - handling of unfinished measurements (mdf 4): warnings are logged based on the unfinished status flags but no further steps are taken to sanitize the measurement
  - full support for remaining mdf 4 channel arrays types
  - xml schema for MDBLOCK: most metadata stored in the comment blocks will not be available
  - full handling of event blocks: events are transfered to the new files (in case of calling methods that return new *MDF* objects) but no new events can be created
  - channels with default X axis: the defaukt X axis is ignored and the channel group's master channel is used

1.4 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
- pandas : for DataFrame export
- canmatrix : to handle CAN bus logging measurements

optional dependencies needed for exports

- h5py : for HDF5 export
- xlsxwriter : for Excel export
- scipy : for Matlab v4 and v5 .mat export
- hdf5storage : for Matlab v7.3 .mat export
- fastparquet : for parquet export

other optional dependencies

- chardet : to detect non-standard unicode encodings
- PyQt4 or PyQt5 : for GUI tool
- pyqtgraph : for GUI tool

1.5 Installation

asammdf is available on

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

1.6 Contributing

Please have a look over the [contributing guidelines](https://github.com/danielhrisca/asammdf/blob/master/CONTRIBUTING.md)

1.6.1 Contributors

Thanks to all who contributed with commits to *asammdf*

- Julien Grave JulienGrv.
- Jed Frey jed-frey.
- Mihai yahym.
- Jack Weinstein jacklev.
- Isuru Fernando isuruf.
- Felix Kohlgrüber fkohlgrueber.
- Stanislav Frolov stanifrolov.
- Thomas Kastl **'kasuteru <https://github.com/kasuteru'_**.
- venden **'venden <https://github.com/venden'_**.

CHAPTER 2

API

2.1 MDF

This class acts as a proxy for the *MDF2*, *MDF3* and *MDF4* classes. All attribute access is delegated to the underlying *_mdf* attribute (MDF2, MDF3 or MDF4 object). See MDF3 and MDF4 for available extra methods (MDF2 and MDF3 share the same implementation).

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.

The 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'*, ***kwargs*)

Unified access to MDF v3 and v4 files. Underlying _mdf's attributes and methods are linked to the *MDF* object via *setattr*. This is done to expose them to the user code and for performance considerations.

Parameters

name [string | BytesIO] mdf file name (if provided it must be a real file name) or file-like object

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', '2.10', '2.14', '3.00', '3.10', '3.20', '3.30', '4.00', '4.10', '4.11'); default '4.10'

callback [function] keyword only argument: function to call to update the progress; the function must accept two arguments (the current progress and maximum progress value)

use_display_names [bool] keyword only argument: for MDF4 files parse the XML channel comment to search for the display name; XML parsing is quite expensive so setting this to *False* can decrease the loading times very much; default *False*

static concatenate (*files*, *version='4.10'*, *memory='full'*, *sync=True*, ***kwargs*)

concatenates several files. 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 or *MDF* instances

version [str] merged file version

memory [str] memory option; default *full*

sync [bool] sync the files based on the start of measurement, default *True*

Returns

concatenate [MDF] new *MDF* object with concatenated channels

Raises

MdfException [if there are inconsistencies between the files]

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

convert *MDF* to other version

Parameters

version [str] new mdf file version from ('2.00', '2.10', '2.14', '3.00', '3.10', '3.20', '3.30', '4.00', '4.10', '4.11'); default '4.10'

memory [str] memory option; default *full*

Returns

out [MDF] new *MDF* object

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

cut *MDF* file. *start* and *stop* limits are absolute values or values relative to the first timestamp depending on the *whence* argument.

Parameters

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

stop [float] stop time, default *None*. 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

version [str] new mdf file version from ('2.00', '2.10', '2.14', '3.00', '3.10', '3.20', '3.30', '4.00', '4.10', '4.11'); default *None* and in this case the original file version is used

memory [str] memory option; default *None* and in this case the original file memory option is used

Returns

out [MDF] new MDF object

export (*fmt*, *filename=None*, ***kargs*)

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 4, 5 or 7.3 export. If *single_time_base==False* the channels will be renamed in the mat file 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)
- *pandas* : export all channels as a single pandas DataFrame

filename [string] export file name

****kwargs**

- *single_time_base*: resample all channels to common time base, default *False* (pandas export is by default single based)
- *raster*: float time raster for resampling. Valid if *single_time_base* is *True* and for *pandas* export
- *time_from_zero*: adjust time channel to start from 0
- *use_display_names*: use display name instead of standard channel name, if available.
- *empty_channels*: behaviour for channels without samples; the options are *skip* or *zeros*; default is *zeros*
- *format*: only valid for *mat* export; can be '4', '5' or '7.3', default is '5'

Returns

dataframe [pandas.DataFrame] only in case of *pandas* export

filter (*channels*, *memory=None*, *version=None*)

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

Parameters

channels [list] list of items to be filtered; each item can be :

- a channel name string
- (channel name, group index, channel index) list or tuple
- (channel name, group index) list or tuple
- (None, group index, channel index) list or tuple

version [str] new mdf file version from ('2.00', '2.10', '2.14', '3.00', '3.10', '3.20', '3.30', '4.00', '4.10', '4.11'); default *None* and in this case the original file version is used

memory [str] memory option; default *None* and in this case the original file memory option is used

Returns

mdf [MDF] new *MDF* file

Examples

```
>>> from asammdf import MDF, Signal
>>> import numpy as np
>>> t = np.arange(5)
>>> s = np.ones(5)
>>> mdf = MDF()
>>> for i in range(4):
...     sigs = [Signal(s*(i*10+j), t, name='SIG') for j in range(1,4)]
...     mdf.append(sigs)
...
>>> filtered = mdf.filter(['SIG', ('SIG', 3, 1), ['SIG', 2], (None, 1, 2)])
>>> for gp_nr, ch_nr in filtered.channels_db['SIG']:
...     print(filtered.get(group=gp_nr, index=ch_nr))
...
<Signal SIG:
        samples=[ 1.  1.  1.  1.  1.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
<Signal SIG:
        samples=[ 31.  31.  31.  31.  31.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
<Signal SIG:
        samples=[ 21.  21.  21.  21.  21.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
<Signal SIG:
        samples=[ 12.  12.  12.  12.  12.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
```

iter_channels(*skip_master=True*)

generator that yields a *Signal* for each non-master channel

Parameters

skip_master [bool] do not yield master channels; default *True*

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

iterator over a channel

This is usefull in case of large files with a small number of channels.

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

raw [bool] return channel samples without appling the conversion rule; default *False*

iter_groups()
generator that yields channel groups as pandas DataFrames

static merge(*files*, *version='4.10'*, *memory='full'*, *sync=True*, ***kwargs*)
concatenates several files. 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 or *MDF* instances

version [str] merged file version

memory [str] memory option; default *full*

sync [bool] sync the files based on the start of measurement, default *True*

Returns

concatenate [MDF] new *MDF* object with concatenated channels

Raises

MdfException [if there are inconsistencies between the files]

resample(*raster*, *memory=None*, *version=None*)
resample all channels using the given raster

Parameters

raster [float] time raster is seconds

version [str] new mdf file version from ('2.00', '2.10', '2.14', '3.00', '3.10', '3.20', '3.30', '4.00', '4.10', '4.11'); default *None* and in this case the original file version is used

memory [str] memory option; default *None* and in this case the original file memory option is used

Returns

mdf [MDF] new *MDF* with resampled channels

static scramble(*name*, *memory='low'*)
scramble text blocks and keep original file strcuture

Parameters

name [str] file name

memory [str] memory option; default *'low'*

select(*channels*, *dataframe=False*)
retreiv the channels listed in *channels* argument as *Signal* objects

Parameters

channels [list] list of items to be filtered; each item can be :

- a channel name string
- (channel name, group index, channel index) list or tuple
- (channel name, group index) list or tuple
- (None, group index, channel index) lsit or tuple

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

Examples

```
>>> from asammdf import MDF, Signal
>>> import numpy as np
>>> t = np.arange(5)
>>> s = np.ones(5)
>>> mdf = MDF()
>>> for i in range(4):
...     sigs = [Signal(s*(i*10+j), t, name='SIG') for j in range(1,4)]
...     mdf.append(sigs)
...
>>> # select SIG group 0 default index 1 default, SIG group 3 index 1, SIG␣
↪group 2 index 1 default and channel index 2 from group 1
...
>>> mdf.select(['SIG', ('SIG', 3, 1), ['SIG', 2],  (None, 1, 2)])
[<Signal SIG:
        samples=[ 1.  1.  1.  1.  1.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
, <Signal SIG:
        samples=[ 31.  31.  31.  31.  31.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
, <Signal SIG:
        samples=[ 21.  21.  21.  21.  21.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
, <Signal SIG:
        samples=[ 12.  12.  12.  12.  12.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
]
```

static stack (*files*, *version='4.10'*, *memory='full'*, *sync=True*, ***kwargs*)
merge several files and return the merged *MDF* object

Parameters

files [list | tuple] list of *MDF* file names or *MDF* instances

version [str] merged file version

memory [str] memory option; default *full*

sync [bool] sync the files based on the start of measurement, default *True*

Returns

merged [MDF] new *MDF* object with merge channels

whereis (*channel*)
get ocurrences of channel name in the file

Parameters

channel [str] channel name string

Returns

ocurrences [tuple]

Examples

```
>>> mdf = MDF(file_name)
>>> mdf.whereis('VehicleSpeed') # "VehicleSpeed" exists in the file
((1, 2), (2, 4))
>>> mdf.whereis('VehicleSPD') # "VehicleSPD" doesn't exist in the file
()
```

2.2 MDF3

class asammdf.mdf_v3.**MDF3** (*name=None*, *memory='full'*, *version='3.30'*, ***kwargs*)
The *header* attibute is a *HeaderBlock*.

The *groups* attribute is a list of dicts, each one with the following keys

- `data_group` - DataGroup object
- `channel_group` - ChannelGroup object
- `channels` - list of Channel objects (when *memory* is *full* or *low*) or addresses (when *memory* is *minimum*) with the same order as found in the mdf file
- `channel_dependencies` - list of *ChannelArrayBlock* in case of channel arrays; list of Channel objects (when *memory* is *full* or *low*) or addresses (when *memory* is *minimum*) in case of structure channel composition
- `data_block` - DataBlock object when *memory* is *full* else address of data block
- `data_location`- integer code for data location (original file, temporary file or memory)
- `data_block_addr` - list of raw samples starting addresses, for *low* and *minimum* memory options
- `data_block_type` - list of codes for data block type

- `data_block_size` - list of raw samples block size
- `sorted` - sorted indicator flag
- `record_size` - dict that maps record ID's to record sizes in bytes
- `size` - total size of data block for the current group
- `trigger` - *Trigger* object for current group

Parameters

name [string] mdf file name (if provided it must be a real file name) or file-like object

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 ('2.00', '2.10', '2.14', '3.00', '3.10', '3.20' or '3.30'); default '3.30'

Attributes

attachments [list] list of file attachments

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

groups [list] list of data group dicts

header [HeaderBlock] mdf file header

identification [FileIdentificationBlock] mdf file start block

masters_db [dict]

used for fast master channel access; for each group index key the value is the master channel index

memory [str] memory optimization option

name [string] mdf file name

version [str] mdf version

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*, *units=None*)
Appends a new data group.

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

Parameters

signals [list | Signal | pandas.DataFrame] list of *Signal* objects, or a single *Signal* object, or a pandas *DataFrame* object

acquisition_info [str] acquisition information; default 'Python'

common_timebase [bool] flag to hint that the signals have the same timebase

units [dict] will contain the signal units mapped to the singal names when appending a pandas DataFrame

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, timstamps=t, unit='+', name='Positive')
>>> s2 = Signal(samples=s2, timstamps=t, unit='-', name='Negative')
>>> s3 = Signal(samples=s3, timstamps=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')
>>> df = pd.DataFrame.from_dict({'s1': np.array([1, 2, 3, 4, 5]), 's2': np.
↪array([-1, -2, -3, -4, -5])})
>>> units = {'s1': 'V', 's2': 'A'}
>>> mdf2.append(df, units=units)
```

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

configure(*read_fragment_size=None*, *write_fragment_size=None*, *use_display_names=None*, *single_bit_uint_as_bool=None*)
configure read and write fragment size for chuncked data access

Parameters

read_fragment_size [int] size hint of splitted data blocks, default 8MB; if the initial size is smaller, then no data list is used. The actual split size depends on the data groups' records size

write_fragment_size [int] size hint of splitted data blocks, default 8MB; if the initial size is smaller, then no data list is used. The actual split size depends on the data groups' records size

extend(*index*, *signals*)
Extend a group with new samples. The first signal is the master channel's samples, and the next signals

must respect the same order in which they were appended. The samples must have raw or physical values according to the *Signals* used for the initial append.

Parameters

index [int] group index

signals [list] list on numpy.ndarray objects

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']
>>> s1 = Signal(samples=s1, timstamps=t, unit='+', name='Positive')
>>> s2 = Signal(samples=s2, timstamps=t, unit='-', name='Negative')
>>> s3 = Signal(samples=s3, timstamps=t, unit='flts', name='Floats')
>>> mdf = MDF3('new.mdf')
>>> mdf.append([s1, s2, s3], 'created by asammdf v1.1.0')
>>> t = np.array([0.006, 0.007, 0.008, 0.009, 0.010])
>>> mdf2.extend(0, [t, s1, s2, s3])
```

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

data [bytes] prevent redundant data read by providing the raw data group samples

raw [bool] return channel samples without appling the conversion rule; default *False*

ignore_invalidation_bits [bool] only defined to have the same API with the MDF v4

Returns

res [(numpy.array, None) | Signal] returns *Signal* if *samples_only*=*False* (default option), otherwise returns a (numpy.array, None) tuple (for compatibility with MDF v4 class.

The *Signal* samples are

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

Raises

MdfException :

*** if the channel name is not found**

*** if the group index is out of range**

*** if the channel index is out of range**

Examples

```
>>> from asammdf import MDF, Signal
>>> import numpy as np
>>> t = np.arange(5)
>>> s = np.ones(5)
>>> mdf = MDF(version='3.30')
>>> for i in range(4):
...     sigs = [Signal(s*(i*10+j), t, name='Sig') for j in range(1, 4)]
...     mdf.append(sigs)
...
>>> # first group and channel index of the specified channel name
...
>>> mdf.get('Sig')
UserWarning: Multiple occurances for channel "Sig". Using first occurance␣
↪from data group 4. Provide both "group" and "index" arguments to select␣
↪another data group
<Signal Sig:
        samples=[ 1.  1.  1.  1.  1.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
>>> # first channel index in the specified group
...
>>> mdf.get('Sig', 1)
<Signal Sig:
        samples=[ 11.  11.  11.  11.  11.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
>>> # channel named Sig from group 1 channel index 2
...
>>> mdf.get('Sig', 1, 2)
<Signal Sig:
        samples=[ 12.  12.  12.  12.  12.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
```

(continues on next page)

(continued from previous page)

```
        comment="">
>>> # channel index 1 or group 2
...
>>> mdf.get(None, 2, 1)
<Signal Sig:
        samples=[ 21.  21.  21.  21.  21.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
>>> mdf.get(group=2, index=1)
<Signal Sig:
        samples=[ 21.  21.  21.  21.  21.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
```

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 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

Returns

comment [str] found channel comment

get_channel_name(*group*, *index*)

Gets channel name.

Parameters

group [int] 0-based group index

index [int] 0-based channel index

Returns

name [str] found channel name

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 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

Returns

unit [str] found channel unit

get_master(*index*, *data=None*, *raster=None*)
returns master channel samples for given group

Parameters

index [int] group index

data [(bytes, int)] (data block raw bytes, fragment offset); default None

raster [float] raster to be used for interpolation; 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=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 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

Returns

output_file [str] output file name

2.2.1 MDF version 2 & 3 blocks

The following classes implement different MDF version3 blocks.

Channel Class

class asammdf.v2_v3_blocks.**Channel**(***kwargs*)
CNBLOCK class

If the *load_metadata* keyword argument is not provided or is False, then the conversion, source and display name information is not processed.

Channel has the following key-value pairs

- `id` - bytes : block ID; always b'CN'
- `block_len` - int : block bytes size
- `next_ch_addr` - int : next CNBLOCK address
- `conversion_addr` - int : address of channel conversion block
- `source_addr` - int : address of channel source block
- `ch_depend_addr` - int : address of dependency block (CDBLOCK) of this channel
- `comment_addr` - int : address of TXBLOCK that contains the channel comment
- `channel_type` - int : integer code for channel type
- `short_name` - bytes : short signal name
- `description` - bytes : signal description
- `start_offset` - int : start offset in bits to determine the first bit of the signal in the data record
- `bit_count` - int : channel bit count
- `data_type` - int : integer code for channel data type
- `range_flag` - int : value range valid flag
- `min_raw_value` - float : min raw value of all samples
- `max_raw_value` - float : max raw value of all samples
- `sampling_rate` - float : sampling rate in *'s'* for a virtual time channel

- `long_name_addr` - int : address of TXBLOCK that contains the channel's name
- `display_name_addr` - int : address of TXBLOCK that contains the channel's display name
- `aditional_byte_offset` - int : additional Byte offset of the channel in the data recor

Parameters

address [int] block address; to be used for objects created from file

stream [handle] file handle; to be used for objects created from file

load_metadata [bool] option to load conversion, source and display_name; default *True*

for dynamically created objects : see the key-value pairs

Examples

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

Attributes

address [int] block address inside mdf file

comment [str] channel comment

conversion [ChannelConversion] channel conversion; *None* if the channel has no conversion

display_name [str] channel display name

name [str] full channel name

source [SourceInformation] channel source information; *None* if the channel has no source information

ChannelConversion Class

class asammdf.v2_v3_blocks.**ChannelConversion**(***kwargs*)

CCBLOCK class

The ChannelConversion object can be created in two modes:

ChannelConversion has the following common key-value pairs

- `id` - bytes : block ID; always b'CC'
- `block_len` - int : block bytes size
- `range_flag` - int : value range valid flag
- `min_phy_value` - float : min raw value of all samples
- `max_phy_value` - float : max raw value of all samples
- `unit` - bytes : physical unit

- `conversion_type` - int : integer code for conversion type
- `ref_param_nr` - int : number of referenced parameters

ChannelConversion has the following specific key-value pairs

- linear conversion
  - `a` - float : factor
  - `b` - float : offset
  - `CANapeHiddenExtra` - bytes : sometimes CANape appends extra information; not compliant with MDF specs
- algebraic conversion
  - `formula` - bytes : ecuation as string
- polynomial or rational conversion
  - `P1` to `P6` - float : parameters
- exponential or logarithmic conversion
  - `P1` to `P7` - float : parameters
- tabular with or without interpolation (grouped by index)
  - `raw_<N>` - int : N-th raw value (X axis)
  - `phys_<N>` - float : N-th physical value (Y axis)
- text table conversion
  - `param_val_<N>` - int : N-th raw value (X axis)
  - `text_<N>` - N-th text physical value (Y axis)
- text range table conversion
  - `default_lower` - float : default lower raw value
  - `default_upper` - float : default upper raw value
  - `default_addr` - int : address of default text physical value
  - `lower_<N>` - float : N-th lower raw value
  - `upper_<N>` - float : N-th upper raw value
  - `text_<N>` - int : address of N-th text physical value

Parameters

address [int] block address inside mdf file

raw_bytes [bytes] complete block read from disk

stream [file handle] mdf file handle

for dynamically created objects : see the key-value pairs

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

formula [str] formula string in case of algebraic conversion

referenced_blocks [list] list of CCBLOCK/TXBLOCK referenced by the conversion

unit [str] physical unit

ChannelDependency Class

class asammdf.v2_v3_blocks.**ChannelDependency**(***kwargs*)

CDBLOCK class

ChannelDependency has the following key-value pairs

- `id` - bytes : block ID; always b'CD'
- `block_len` - int : block bytes size
- `dependency_type` - int : integer code for dependency type
- `sd_nr` - int : total number of signals dependencies
- `dg_<N>` - address of data group block (DGBLOCK) of N-th signal dependency
- `dg_<N>` - address of channel group block (CGBLOCK) of N-th signal dependency
- `dg_<N>` - address of channel block (CNBLOCK) of N-th signal dependency
- `dim_<K>` - int : Optional size of dimension *K* for N-dimensional dependency

Parameters

stream [file handle] mdf file handle

address [int] block address inside mdf file

for dynamically created objects : see the key-value pairs

Attributes

address [int] block address inside mdf file

referenced_channels [list] list of (group index, channel index) pairs

ChannelExtension Class

class asammdf.v2_v3_blocks.**ChannelExtension**(***kwargs*)

CEBLOCK class

Channel has the following common key-value pairs

- `id` - bytes : block ID; always b'CE'
- `block_len` - int : block bytes size

- `type` - int : extension type identifier

Channel has the following specific key-value pairs

- for DIM block
  - `module_nr` - int: module number
  - `module_address` - int : module address
  - `description` - bytes : module description
  - `ECU_identification` - bytes : identification of ECU
  - `reserved0` - bytes : reserved bytes
- for Vector CAN block
  - `CAN_id` - int : CAN message ID
  - `CAN_ch_index` - int : index of CAN channel
  - `message_name` - bytes : message name
  - `sender_name` - btyes : sender name
  - `reserved0` - bytes : reserved bytes

Parameters

stream [file handle] mdf file handle

address [int] block address inside mdf file

for dynamically created objects : see the key-value pairs

Attributes

address [int] block address inside mdf file

comment [str] extension comment

name [str] extension name

path [str] extension path

ChannelGroup Class

class asammdf.v2_v3_blocks.**ChannelGroup**(***kwargs*)

CGBLOCK class

ChannelGroup has the following key-value pairs

- `id` - bytes : block ID; always b'CG'
- `block_len` - int : block bytes size
- `next_cg_addr` - int : next CGBLOCK address
- `first_ch_addr` - int : address of first channel block (CNBLOCK)
- `comment_addr` - int : address of TXBLOCK that contains the channel group comment
- `record_id` - int : record ID used as identifier for a record if the DGBLOCK defines a number of record IDs > 0 (unsorted group)
- `ch_nr` - int : number of channels

- samples_byte_nr - int : size of data record in bytes without record ID
- cycles_nr - int : number of cycles (records) of this type in the data block
- sample_reduction_addr - int : addresss to first sample reduction block

Parameters

stream [file handle] mdf file handle

address [int] block address inside mdf file

for dynamically created objects : see the key-value pairs

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

comment [str] channel group comment

DataGroup Class

class asammdf.v2_v3_blocks.**DataGroup**(***kwargs*)

DGBLOCK class

Channel has the following key-value pairs

- id - bytes : block ID; always b'DG'
- block_len - int : block bytes size
- next_dg_addr - int : next DGBLOCK address
- first_cg_addr - int : address of first channel group block (CGBLOCK)
- trigger_addr - int : address of trigger block (TRBLOCK)
- data_block_addr - addrfss of data block
- cg_nr - int : number of channel groups
- record_id_len - int : number of record IDs in the data block
- reserved0 - bytes : reserved bytes

Parameters

stream [file handle] mdf file handle

address [int] block address inside mdf file

for dynamically created objects : see the key-value pairs

Attributes

address [int] block address inside mdf file

FileIdentificationBlock Class

class asammdf.v2_v3_blocks.**FileIdentificationBlock**(***kwargs*)

IDBLOCK class

FileIdentificationBlock has the following key-value pairs

- `file_identification` - bytes : file identifier
- `version_str` - bytes : format identifier
- `program_identification` - bytes : creator program identifier
- `byte_order` - int : integer code for byte order (endiannes)
- `float_format` - int : integer code for floating-point format
- `mdf_version` - int : version number of MDF format
- `code_page` - int : unicode code page number
- `reserved0` - bytes : reserved bytes
- `reserved1` - bytes : reserved bytes
- `unfinalized_standard_flags` - int : standard flags for unfinalized MDF
- `unfinalized_custom_flags` - int : custom flags for unfinalized MDF

Parameters

stream [file handle] mdf file handle

version [int] mdf version in case of new file (dynamically created)

Attributes

address [int] block address inside mdf file; should be 0 always

HeaderBlock Class

class asammdf.v2_v3_blocks.**HeaderBlock**(***kwargs*)

HDBLOCK class

HeaderBlock has the following key-value pairs

- `id` - bytes : block ID; always b'HD'
- `block_len` - int : block bytes size
- `first_dg_addr` - int : address of first data group block (DGBLOCK)
- `comment_addr` - int : address of TXBLOCK taht contains the measurement file comment
- `program_addr` - int : address of program block (PRBLOCK)
- `dg_nr` - int : number of data groups
- `date` - bytes : date at which the recording was started in "DD:MM:YYYY" format
- `time` - btyes : time at which the recording was started in "HH:MM:SS" format

- `author` - btyes : author name
- `organization` - bytes : organization name
- `project` - bytes : project name
- `subject` - bytes : subject

Since version 3.2 the following extra keys were added:

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

Parameters

stream [file handle] mdf file handle

version [int] mdf version in case of new file (dynamically created)

Attributes

address [int] block address inside mdf file; should be 64 always

comment [int] file comment

program [ProgramBlock] program block

author [str] measurement author

department [str] author's department

project [str] working project

subject [str] measurement subject

ProgramBlock Class

class asammdf.v2_v3_blocks.**ProgramBlock**(***kwargs*)

PRBLOCK class

ProgramBlock has the following key-value pairs

- `id` - bytes : block ID; always b'PR'
- `block_len` - int : block bytes size
- `data` - btyes : creator program free format 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.v2_v3_blocks.**SampleReduction**(**kwargs)
SRBLOCK class

SampleReduction has the following key-value pairs

- `id` - bytes : block ID; always b'SR'
- `block_len` - int : block bytes size
- `next_sr_addr` - int : next SRBLOCK address
- `data_block_addr` - int : address of data block for this sample reduction
- `cycles_nr` - int : number of reduced samples in the data block
- **`time_interval` - float** [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.v2_v3_blocks.**TextBlock**(**kwargs)
TXBLOCK class

TextBlock has the following key-value pairs

- `id` - bytes : block ID; always b'TX'
- `block_len` - int : block bytes size
- `text` - bytes : text content

Parameters

stream [file handle] mdf file handle

address [int] block address inside mdf file

text [bytes | str] bytes or str for creating a new TextBlock

Examples

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

Attributes

address [int] block address inside mdf file

TriggerBlock Class

class asammdf.v2_v3_blocks.**TriggerBlock**(***kwargs*)
TRBLOCK class

Channel has the following key-value pairs

- `id` - bytes : block ID; always b'TR'
- `block_len` - int : block bytes size
- `text_addr` - int : address of TXBLOCK that contains the trigger comment text
- `trigger_events_nr` - int : number of trigger events
- `trigger_<N>_time` - float : trigger time [s] of trigger's N-th event
- `trigger_<N>_pretime` - float : pre trigger time [s] of trigger's N-th event
- `trigger_<N>_posttime` - float : post trigger time [s] of trigger's N-th event

Parameters

stream [file handle] mdf file handle

address [int] block address inside mdf file

Attributes

address [int] block address inside mdf file

comment [str] trigger comment

2.3 MDF4

class asammdf.mdf_v4.**MDF4**(*name=None*, *memory='full'*, *version='4.10'*, ***kwargs*)
The *header* attibute is a *HeaderBlock*.

The *groups* attribute is a list of dicts, each one with the following keys:

- `data_group` - DataGroup object
- `channel_group` - ChannelGroup object
- `channels` - list of Channel objects (when *memory* is *full* or *low*) or addresses (when *memory* is *minimum*) with the same order as found in the mdf file
- `channel_dependencies` - list of *ChannelArrayBlock* in case of channel arrays; list of Channel objects (when *memory* is *full* or *low*) or addresses (when *memory* is *minimum*) in case of structure channel composition
- `data_block` - DataBlock object when *memory* is *full* else address of data block
- `data_location`- integer code for data location (original file, temporary file or memory)
- `data_block_addr` - list of raw samples starting addresses, for *low* and *minimum* memory options
- `data_block_type` - list of codes for data block type
- `data_block_size` - list of raw samples block size
- `sorted` - sorted indicator flag
- `record_size` - dict that maps record ID's to record sizes in bytes (including invalidation bytes)

- `param` - row size used for tranposizition, in case of tranposed zipped blockss

Parameters

name [string] mdf file name (if provided it must be a real file name) or file-like object

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'

callback [function] keyword only argument: function to call to update the progress; the function must accept two arguments (the current progress and maximum progress value)

use_display_names [bool] keyword only argument: for MDF4 files parse the XML channel comment to search for the display name; XML parsing is quite expensive so setting this to *False* can decrease the loading times very much; default *False*

Attributes

attachments [list] list of file attachments

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

events [list] list event blocks

file_comment [TextBlock] file comment TextBlock

file_history [list] list of (FileHistory, TextBlock) pairs

groups [list] list of data group dicts

header [HeaderBlock] mdf file header

identification [FileIdentificationBlock] mdf file start block

masters_db [dict]

used for fast master channel access; for each group index key the value is the master channel index

memory [str] memory optimization option

name [string] mdf file name

version [str] mdf version

append(*signals*, *source_info='Python'*, *common_timebase=False*, *units=None*)

Appends a new data group.

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

Parameters

signals [list | Signal | pandas.DataFrame] list of *Signal* objects, or a single *Signal* object, or a pandas *DataFrame* object

source_info [str] source information; default 'Python'

common_timebase [bool] flag to hint that the signals have the same timebase

units [dict] will contain the signal units mapped to the singal names when appending a pandas DataFrame

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, timstamps=t, unit='+', name='Positive')
>>> s2 = Signal(samples=s2, timstamps=t, unit='-', name='Negative')
>>> s3 = Signal(samples=s3, timstamps=t, unit='flts', name='Floats')
>>> mdf = MDF4('new.mdf')
>>> mdf.append([s1, s2, s3], 'created by asammdf v4.0.0')
>>> # 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 v4.0.0')
>>> mdf2.append(ch1, 'just a single channel')
>>> df = pd.DataFrame.from_dict({'s1': np.array([1, 2, 3, 4, 5]), 's2': np.
↪array([-1, -2, -3, -4, -5])})
>>> units = {'s1': 'V', 's2': 'A'}
>>> mdf2.append(df, units=units)
```

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

Returns

index [int] new attachment index

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

configure(*read_fragment_size=None*, *write_fragment_size=None*, *use_display_names=None*, *single_bit_uint_as_bool=None*)

configure read and write fragment size for chuncked data access

Parameters

read_fragment_size [int] size hint of splitted data blocks, default 8MB; if the initial size is smaller, then no data list is used. The actual split size depends on the data groups' records size

write_fragment_size [int] size hint of splitted data blocks, default 8MB; if the initial size is smaller, then no data list is used. The actual split size depends on the data groups' records size

extend(*index*, *signals*)

Extend a group with new samples. The first signal is the master channel's samples, and the next signals must respect the same order in which they were appended. The samples must have raw or physical values according to the *Signals* used for the initial append.

Parameters

index [int] group index

signals [list] list on numpy.ndarray objects

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']
>>> s1 = Signal(samples=s1, timstamps=t, unit='+', name='Positive')
>>> s2 = Signal(samples=s2, timstamps=t, unit='-', name='Negative')
>>> s3 = Signal(samples=s3, timstamps=t, unit='flts', name='Floats')
>>> mdf = MDF3('new.mdf')
>>> mdf.append([s1, s2, s3], 'created by asammdf v1.1.0')
>>> t = np.array([0.006, 0.007, 0.008, 0.009, 0.010])
>>> mdf2.extend(0, [t, s1, s2, s3])
```

extract_attachment(*address=None*, *index=None*)

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

Parameters

address [int] attachment index; default *None*

index [int] attachment index; default *None*

Returns

data [bytes | str] attachment data

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

data [bytes] prevent redundant data read by providing the raw data group samples

raw [bool] return channel samples without appling the conversion rule; default *False*

ignore_invalidation_bits [bool] option to ignore invalidation bits

Returns

res [(numpy.array, numpy.array) | Signal] returns *Signal* if *samples_only*=*False* (default option), otherwise returns a (numpy.array, numpy.array) tuple of samples and invalidation bits. If invalidation bits are not used or if *ignore_invalidation_bits* if False, then the second item will be None.

The *Signal* samples are:

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

Raises

MdfException :

*** if the channel name is not found**

*** if the group index is out of range**

*** if the channel index is out of range**

Examples

```
>>> from asammdf import MDF, Signal
>>> import numpy as np
>>> t = np.arange(5)
>>> s = np.ones(5)
>>> mdf = MDF(version='4.10')
>>> for i in range(4):
...     sigs = [Signal(s*(i*10+j), t, name='Sig') for j in range(1, 4)]
...     mdf.append(sigs)
...
>>> # first group and channel index of the specified channel name
...
```

(continues on next page)

(continued from previous page)

```
>>> mdf.get('Sig')
UserWarning: Multiple occurances for channel "Sig". Using first occurance␣
↪from data group 4. Provide both "group" and "index" arguments to select␣
↪another data group
<Signal Sig:
        samples=[ 1.  1.  1.  1.  1.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">ref_ch_nr, ref_dg_nr = ca_block.referenced_channels[i]
>>> # first channel index in the specified group
...
>>> mdf.get('Sig', 1)
<Signal Sig:
        samples=[ 11.  11.  11.  11.  11.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
>>> # channel named Sig from group 1 channel index 2
...
>>> mdf.get('Sig', 1, 2)
<Signal Sig:
        samples=[ 12.  12.  12.  12.  12.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
>>> # channel index 1 or group 2
...
>>> mdf.get(None, 2, 1)
<Signal Sig:
        samples=[ 21.  21.  21.  21.  21.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
>>> mdf.get(group=2, index=1)
<Signal Sig:
        samples=[ 21.  21.  21.  21.  21.]
        timestamps=[0 1 2 3 4]
        unit=""
        info=None
        comment="">
```

get_can_signal (*name*, *database=None*, *db=None*, *ignore_invalidation_bits=False*)

get CAN message signal. You can specify an external CAN database (*database* argument) or canmatrix databse object that has already been loaded from a file (*db* argument).

The signal name can be specified in the following ways

- `CAN<ID>.<MESSAGE_NAME>.<SIGNAL_NAME>` - the *ID* value starts from 1 and must match the ID found in the measurement (the source CAN bus ID) Example: CAN1.Wheels.FL_WheelSpeed
- `CAN<ID>.CAN_DataFrame_<MESSAGE_ID>.<SIGNAL_NAME>` - the *ID* value starts from 1 and the *MESSAGE_ID* is the decimal message ID as found in the database. Example: CAN1.CAN_DataFrame_218.FL_WheelSpeed

- `<MESSAGE_NAME>.SIGNAL_NAME` - in this case the first occurence of the message name and signal are returned (the same message could be found on muplit CAN buses; for example on CAN1 and CAN3) Example: Wheels.FL_WheelSpeed
- `CAN_DataFrame_<MESSAGE_ID>.<SIGNAL_NAME>` - in this case the first occurence of the message name and signal are returned (the same message could be found on muplit CAN buses; for example on CAN1 and CAN3). Example: CAN_DataFrame_218.FL_WheelSpeed
- `<SIGNAL_NAME>` - in this case the first occurence of the signal name is returned (the same signal anme coudl be found in multiple messages and on multiple CAN buses). Example: FL_WheelSpeed

Parameters

name [str] signal name

database [str] path of external CAN database file (.dbc or .arxml); default *None*

db [canmatrix.database] canmatrix CAN database object; default *None*

ignore_invalidation_bits [bool] option to ignore invalidation bits

Returns

sig [Signal] Signal object with the physical values

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_name(*group*, *index*)

Gets channel name.

Parameters

group [int] 0-based group index

index [int] 0-based channel index

Returns

name [str] found channel name

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_invalidation_bits(*group_index*, *channel*, *fragment*)
get invalidation indexes for the channel

Parameters

group_index [int] group index

channel [Channel] channel object

fragment [(bytes, int)] (fragment bytes, fragment offset)

Returns

invalidation_bits [iterable] iterable of valid channel indexes; if all are valid *None* is returned

get_master(*index*, *data=None*, *raster=None*)
returns master channel samples for given group

Parameters

index [int] group index

data [(bytes, int)] (data block raw bytes, fragment offset); default None

raster [float] raster to be used for interpolation; 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=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 '_<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)

Returns

output_file [str] output file name

2.3.1 MDF version 4 blocks

The following classes implement different MDF version4 blocks.

AttachmentBlock Class

class asammdf.v4_blocks.**AttachmentBlock**(***kwargs*)

ATBLOCK class

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

AttachmentBlock has the following key-value pairs

- `id` - bytes : block ID; always b'##AT'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `next_at_addr` - int : next ATBLOCK address
- `file_name_addr` - int : address of TXBLOCK that contains the attachment file name
- `mime_addr` - int : address of TXBLOCK that contains the attachment mime type description
- `comment_addr` - int : address of TXBLOCK/MDBLOCK that contains the attachment comment
- `flags` - int : ATBLOCK flags
- `creator_index` - int : index of file history block
- `reserved1` - int : reserved bytes

- `md5_sum` - bytes : attachment file md5 sum
- `original_size` - int : original uncompress file size in bytes
- `embedded_size` - int : embedded compressed file size in bytes
- `embedded_data` - bytes : embedded atatchment bytes

Parameters

address [int] block address; to be used for objects created from file

stream [handle] file handle; to be used for objects created from file

for dynamically created objects : see the key-value pairs

Attributes

address [int] attachment address

file_name [str] attachment file name

mime [str] mime type

comment [str] attachment comment

Channel Class

class asammdf.v4_blocks.**Channel**(***kwargs*)

CNBLOCK class

If the *load_metadata* keyword argument is not provided or is False, then the conversion, source and display name information is not processed. Further more if the *parse_xml_comment* is not provided or is False, then the display name information from the channel comment is not processed (this is done to avoid expensive XML operations)

Channel has the following key-value pairs

- `id` - bytes : block ID; always b'##CN'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `next_ch_addr` - int : next ATBLOCK address
- `component_addr` - int : address of first channel in case of structure channel composition, or ChannelArrayBlock in case of arrays file name
- `name_addr` - int : address of TXBLOCK that contains the channel name
- `source_addr` - int : address of channel source block
- `conversion_addr` - int : address of channel conversion block
- `data_block_addr` - int : address of signal data block for VLSD channels
- `unit_addr` - int : address of TXBLOCK that contains the channel unit
- `comment_addr` - int : address of TXBLOCK/MDBLOCK that contains the channel comment
- `attachment_<N>_addr` - int : address of N-th ATBLOCK referenced by the current channel; if no ATBLOCK is referenced there will be no such key-value pair

- `default_X_dg_addr` - int : address of DGBLOCK where the default X axis channel for the current channel is found; this key-value pair will not exist for channels that don't have a default X axis
- `default_X_cg_addr` - int : address of CGBLOCK where the default X axis channel for the current channel is found; this key-value pair will not exist for channels that don't have a default X axis
- `default_X_ch_addr` - int : address of default X axis channel for the current channel; this key-value pair will not exist for channels that don't have a default X axis
- `channel_type` - int : integer code for the channel type
- `sync_type` - int : integer code for the channel's sync type
- `data_type` - int : integer code for the channel's data type
- `bit_offset` - int : bit offset
- `byte_offset` - int : byte offset within the data record
- `bit_count` - int : channel bit count
- `flags` - int : CNBLOCK flags
- `pos_invalidation_bit` - int : invalidation bit position for the current channel if there are invalidation bytes in the data record
- `precision` - int : integer code for teh precision
- `reserved1` - int : reserved bytes
- `min_raw_value` - int : min raw value of all samples
- `max_raw_value` - int : max raw value of all samples
- `lower_limit` - int : min physical value of all samples
- `upper_limit` - int : max physical value of all samples
- `lower_ext_limit` - int : min physical value of all samples
- `upper_ext_limit` - int : max physical value of all samples

Parameters

address [int] block address; to be used for objects created from file

stream [handle] file handle; to be used for objects created from file

load_metadata [bool] option to load conversion, source and display_name; default *True*

parse_xml_comment [bool] option to parse XML channel comment to search for display name; default *True*

for dynamically created objects : see the key-value pairs

Attributes

address [int] channel address

attachments [list] list of referenced attachment blocks indexes; the index referece to the attachment block index

comment [str] channel comment

conversion [ChannelConversion] channel conversion; *None* if the channel has no conversion

display_name [str] channel display name; this is extracted from the XML channel comment

name [str] channel name

source [SourceInformation] channel source information; *None* if the channel has no source information

unit [str] channel unit

ChannelConversion Class

class asammdf.v4_blocks.**ChannelConversion**(***kwargs*)

CCBLOCK class

ChannelConversion has the following common key-value pairs

- `id` - bytes : block ID; always b'##CG'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `name_addr` - int : address of TXBLOCK that contains the conversion name
- `unit_addr` - int : address of TXBLOCK that contains the conversion unit
- `comment_addr` - int : address of TXBLOCK/MDBLOCK that contains the conversion comment
- `inv_conv_addr` int : address of invers conversion
- `conversion_type` int : integer code for conversion type
- `precision` - int : integer code for precision
- `flags` - int : conversion block flags
- `ref_param_nr` - int : number fo referenced parameters (linked parameters)
- `val_param_nr` - int : number of value parameters
- `min_phy_value` - float : minimum physical channel value
- `max_phy_value` - float : maximum physical channel value

ChannelConversion has the following specific key-value pairs

- linear conversion
  - `a` - float : factor
  - `b` - float : offset
- rational conversion
  - `P1` to `P6` - float : parameters
- algebraic conversion
  - `formula_add` - address of TXBLOCK that contains the the algebraic conversion formula
- tabluar conversion with or without interpolation
  - `raw_<N>` - float : N-th raw value
  - `phys_<N>` - float : N-th physical value
- tabular range conversion
  - `lower_<N>` - float : N-th lower value
  - `upper_<N>` - float : N-th upper value

  - phys_<N> - float : N-th physical value
- tabular value to text conversion
  - val_<N> - float : N-th raw value
  - text_<N> - int : address of N-th TXBLOCK that contains the physical value
  - default - int : address of TXBLOCK that contains the default physical value
- tabular range to text conversion
  - lower_<N> - float : N-th lower value
  - upper_<N> - float : N-th upper value
  - text_<N> - int : address of N-th TXBLOCK that contains the physical value
  - default - int : address of TXBLOCK that contains the default physical value
- text to value conversion
  - val_<N> - float : N-th physical value
  - text_<N> - int : address of N-th TXBLOCK that contains the raw value
  - val_default - float : default physical value
- text tranfosrmation (translation) conversion
  - input_<N>_addr - int : address of N-th TXBLOCK that contains the raw value
  - output_<N>_addr - int : address of N-th TXBLOCK that contains the physical value
  - default_addr - int : address of TXBLOCK that contains the default physical value

Attributes

address [int] channel conversion address

comment [str] channel conversion comment

formula [str] algebraic conversion formula; default ''

referenced_blocks [list] list of refenced blocks; can be TextBlock objects for value to text, and text to text conversions; for partial conversions the referenced blocks can be ChannelConversion obejct as well

name [str] channel conversion name

unit [str] channel conversion unit

ChannelGroup Class

class asammdf.v4_blocks.**ChannelGroup**(***kwargs*)

CGBLOCK class

ChannelGroup has the following key-value pairs

- id - bytes : block ID; always b'##CG'
- reserved0 - int : reserved bytes
- block_len - int : block bytes size
- links_nr - int : number of links
- next_cg_addr - int : next channel group address

- first_ch_addr - int : address of first channel of this channel group
- acq_name_addr - int : address of TextBLock that contains the channel group acquisition name
- acq_source_addr - int : addres of SourceInformation that contains the channel group source
- first_sample_reduction_addr - int : address of first SRBLOCK; this is considered 0 since sample reduction is not yet supported
- comment_addr - int : address of TXBLOCK/MDBLOCK that contains the channel group comment
- record_id - int : record ID for thei channel group
- cycles_nr - int : number of cycles for this channel group
- flags - int : channel group flags
- path_separator - int : ordinal for character used as path separator
- reserved1 - int : reserved bytes
- samples_byte_nr - int : number of bytes used for channels samples in the record for this channel group; this does not contain the invalidation bytes
- invalidation_bytes_nr - int : number of bytes used for invalidation bits by this channl group

Attributes

acq_name [str] acquisition name

acq_source [SourceInformation] acquisition source information

address [int] channel group address

comment [str] channel group comment

DataGroup Class

class asammdf.v4_blocks.**DataGroup**(***kwargs*)

DGBLOCK class

DataGroup has the following key-value pairs

- id - bytes : block ID; always b'##DG'
- reserved0 - int : reserved bytes
- block_len - int : block bytes size
- links_nr - int : number of links
- next_dg_addr - int : address of next data group block
- first_cg_addr - int : address of first channel group for this data group
- data_block_addr - int : address of DTBLOCK, DZBLOCK, DLBLOCK or HLBLOCK that contains the raw samples for this data group
- comment_addr - int : address of TXBLOCK/MDBLOCK tha contains the data group comment
- record_id_len - int : size of record ID used in case of unsorted data groups; can be 1, 2, 4 or 8
- reserved1 - int : reserved bytes

Attributes

address [int] dat group address

comment [str] data group comment

DataList Class

class asammdf.v4_blocks.**DataList**(****kwargs*)
DLBLOCK class

DataList has the following common key-value pairs

- `id` - bytes : block ID; always b'##DL'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `next_dl_addr` - int : address of next DLBLOCK
- `data_block_addr<N>` - int : address of N-th data block
- `flags` - int : data list flags
- `reserved1` - int : reserved bytes
- `data_block_nr` - int : number of data blocks referenced by thsi list
- for equall lenght blocks
  - `data_block_len` - int : equall uncompressed size in bytes for all referenced data blocks; last block can be smaller
- for variable lenght blocks
  - `offset_<N>` - int : byte offset of N-th data block

Attributes

address [int] data list address

DataBlock Class

class asammdf.v4_blocks.**DataBlock**(****kwargs*)
DTBLOCK class

DataBlock has the following key-value pairs

- `id` - bytes : block ID; always b'##DT'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `data` - bytes : raw samples

Parameters

address [int] DTBLOCK address inside the file

stream [int] file handle

Attributes

address [int] data block address

FileIdentificationBlock Class

class asammdf.v4_blocks.**FileIdentificationBlock**(**kwargs)

IDBLOCK class

FileIdentificationBlock has the following key-value pairs

- `file_identification` - bytes : file identifier
- `version_str` - bytes : format identifier
- `program_identification` - bytes : creator program identifier
- `reserved0` - bytes : reserved bytes
- `mdf_version` - int : version number of MDF format
- `reserved1` - bytes : reserved bytes
- `unfinalized_standard_flags` - int : standard flags for unfinalized MDF
- `unfinalized_custom_flags` - int : custom flags for unfinalized MDF

Attributes

address [int] should always be 0

HeaderBlock Class

class asammdf.v4_blocks.**HeaderBlock**(**kwargs)

HDBLOCK class

HeaderBlock has the following common key-value pairs

- `id` - bytes : block ID; always b'##HD'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `first_dg_addr` - int : address of first DGBLOCK
- `file_history_addr` - int : address of first FHBLOCK
- `channel_tree_addr` - int : address of first CHBLOCK
- `first_attachment_addr` - int : address of first ATBLOCK
- `first_event_addr` - int : address of first EVBLOCK
- `comment_addr` - int : address of TXBLOCK/MDBLOCK that contains the file comment
- `abs_time` - int : time stamp at which recording was started in nanoseconds.
- `tz_offset` - int : UTC time offset in hours (= GMT time zone)
- `daylight_save_time` - int : daylight saving time
- `time_flags` - int : time flags
- `time_quality` - int : time quality flags

- `flags` - int : file flags
- `reserved1` - int : reserved bytes
- `start_angle` - int : angle value at measurement start
- `start_distance` - int : distance value at measurement start

Attributes

address [int] header address

comment [str] file comment

author [str] measurement author

department [str] author's department

project [str] working project

subject [str] measurement subject

start_time

getter and setter the measurement start timestamp

Returns

timestamp [datetime.datetime] start timestamp

SourceInformation Class

class asammdf.v4_blocks.**SourceInformation**(***kwargs*)

SIBLOCK class

SourceInformation has the following key-value pairs

- `id` - bytes : block ID; always b'##SI'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `name_addr` - int : address of TXBLOCK that contains the source name
- `path_addr` - int : address of TXBLOCK that contains the source path
- `comment_addr` - int : address of TXBLOCK/MDBLOCK tha contains the source comment
- `source_type` - int : integer code for source type
- `bus_type` - int : integer code for source bus type
- `flags` - int : source flags
- `reserved1` - bytes : reserved bytes

Attributes

address [int] source information address

comment [str] source comment

name [str] source name

path [str] source path

FileHistory Class

class asammdf.v4_blocks.**FileHistory**(**kwargs)
FHBLOCK class

FileHistory has the following common key-value pairs

- `id` - bytes : block ID; always b'##FH'
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `next_fh_addr` - int : address of next FHBLOCK
- `comment_addr` - int : address of TXBLOCK/MDBLOCK that contains the file history comment
- `abs_time` - int : time stamp at which the file modification happened
- `tz_offset` - int : UTC time offset in hours (= GMT time zone)
- `daylight_save_time` - int : daylight saving time
- `time_flags` - int : time flags
- `reserved1` - bytes : reserved bytes

Attributes

address [int] file history address

comment [str] history comment

TextBlock Class

class asammdf.v4_blocks.**TextBlock**(**kwargs)
common TXBLOCK and MDBLOCK class

TextBlock has the following key-value pairs

- `id` - bytes : block ID; b'##TX' for TXBLOCK and b'##MD' for MDBLOCK
- `reserved0` - int : reserved bytes
- `block_len` - int : block bytes size
- `links_nr` - int : number of links
- `text` - bytes : actual text content

Parameters

address [int] block address

stream [handle] file handle

meta [bool] flag to set the block type to MDBLOCK for dynamically created objects; default *False*

text [bytes/str] text content for dynamically created objects

Attributes

address [int] text block address

2.4 Signal

class asammdf.signal.**Signal**(*samples=None*, *timestamps=None*, *unit=''*, *name=''*, *conversion=None*, *comment=''*, *raw=True*, *master_metadata=None*, *display_name=''*, *attachment=()*, *source=None*, *bit_count=None*, *stream_sync=False*, *invalidation_bits=None*)

The *Signal* represents a channel described by it's samples and timestamps. It can perform arithmetic operations against other *Signal* or numeric types. The operations are computed in respect to the timestamps (time correct). The non-float signals are not interpolated, instead the last value relative to the current timestamp is used. *samples*, *timstamps* 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 | channel conversion block] dict that contains extra conversion information about the signal , default *None*

comment [str] signal comment, default ''

raw [bool] signal samples are raw values, with no physical conversion applied

master_metadata [list] master name and sync type

display_name [str] display name used by mdf version 3

attachment [bytes, name] channel attachment and name from MDF version 4

source [NamedTuple] source information named tuple

bit_count [int] bit count; useful for integer channels

stream_sync [bool] the channel is a synchronisation for the attachment stream (mdf v4 only)

invalidation_bits [numpy.array | None] channel invalidation bits, default *None*

astype(*np_type*)

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

Parameters

np_type [np.dtype] new numpy dtye

Returns

signal [Signal] new *Signal* with the samples of *np_type* dtype

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]

Returns

signal [Signal] new extended *Signal*

interp(*new_timestamps*)
returns a new *Signal* interpolated using the *new_timestamps*

Parameters

new_timestamps [np.array] timestamps used for interpolation

Returns

signal [Signal] new interpolated *Signal*

physical()
get the physical samples values

Returns

phys [Signal] new *Signal* with physical values

plot()
plot Signal samples

CHAPTER 3

Bus logging

Initial read only mode for mdf version 4.10 files containg CAN bus logging is now implemented.

To handle this, **canmatrix** package was added to the dependecies; you will need to install the latest code from the **canmatrix** development branch.

Let's take for example the following situation: the .dbc contains the definition for the CAN message called "VehicleStatus" with ID=123. This message contains the signal "EngineStatus". Logging was made from the CAN bus with ID=1 (CAN1).

There multiple ways to address this channel in this situation:

1. short signal name as found in the .dbc file

```
mdf.get('EngineStatus')
```

2. dbc message name and short signal name, delimited by dot

```
mdf.get('VehicleStatus.EngineStatus')
```

3. CAN bus ID, dbc message name and short signal name, delimited by dot

```
mdf.get('CAN1.VehicleStatus.EngineStatus')
```

4. ASAM conformant message ID and short signal name, delimited by dot

```
mdf.get('CAN_DataFrame_123.EngineStatus')
```

5. CAN bus ID, ASAM conformant message ID and short signal name, delimited by dot

```
mdf.get('CAN1.CAN_DataFrame_123.EngineStatus')
```

CHAPTER 4

Tips

4.1 Impact of *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.

4.1.1 *MDF* created with *memory='full'*

Advantages

- best performance if all channels are used (for example *cut*, *convert*, *export* or *merge* methods)

Disadvantages

- higher RAM usage, there is the chance of MemoryError for large files
- data is not accessed in chunks
- time can be wasted if only a small number of channels is retreived from the file (for example *filter*, *get* or *select* methods)

Use case

- when data fits inside the system RAM

4.1.2 *MDF* created with *memory='low'*

Advantages

- lower RAM usage than memory='full'

- can handle files that do not fit in the available physical memory
- middle ground between 'full' speed and 'minimum' memory usage

Disadvantages

- slower performance for retrieving 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 'full' data exceeds available RAM
- it is advised to avoid getting individual channels when using this option
- best performance / memory usage ratio when using *cut*, *convert*, *filter*, *merge* or *select* methods

Note: See benchmarks for the effects of using the flag

4.1.3 *MDF* created with *memory='minimum'*

Advantages

- lowest RAM usage
- the only choise when dealing with huge files (10's of thousands of channels and GB of sample data)
- handle big files on 32 bit Python ()

Disadvantages

- slightly slower performance compared to momeory='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

4.2 Chunked data access

When the *MDF* is created with the option "full" all the samples are loaded into RAM and are processed as a single block. For large files this can lead to MemoryError exceptions (for example trying to merge several GB sized files).

asammdf optimizes memory usage for options "low" and "minimum" by processing samples in fragments. The read fragment size was tuned based on experimental measurements and should give a good compromise between execution time and memory usage.

You can further tune the read fragment size using the *configure* method, to favor execution speed (using larger fragment sizes) or memory usage (using lower fragment sizes).

4.3 Optimized methods

The *MDF* methods (*cut*, *filter*, *select*) are optimized and should be used instead of calling *get* for several channels. For "low" and "minimum" options the time savings can be dramatic.

4.4 Faster file loading

4.4.1 BytesIO and *memory='full'*

In case of files with high block count (large number of channels, or large number of data blocks) you cand speed up the loading in case of *full* memory option, at the expense of higher RAM usage by reading the file into a *BytesIO* object and feeding it to the *MDF* class

Using a test file with the size of 3.2GB that contained ~580000 channels the loading time and RAM usage were

- Python 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
- Windows-10-10.0.15063-SP0
- Intel64 Family 6 Model 94 Stepping 3, GenuineIntel
- 16GB installed RAM

Open file	Time [ms]	RAM [MB]
asammdf 3.5.1.dev mdfv4	62219	4335
asammdf w BytesIO 3.5.1.dev mdfv4	31232	7409

4.4.2 Skip XML parsing for MDF4 files

MDF4 uses the XML channel comment to define the channel's display name (this acts as an alias for the channel name). XML pasring is an expensive operation that can have a big impact on the loading performance of measurements with hihg channel count.

You can use the keyword only argument *use_display_names* when creating MDF objects to skip the XML parsing. This means that the display names will not be available when calling the *get* method.

Using a test file that contained ~36000 channels the loading times were

Open file	Time [ms]	RAM [MB]
asammdf 3.5.1.dev full mdfv4 use_display_names=True	6086	335
asammdf 3.5.1.dev low mdfv4 use_display_names=True	5590	170
asammdf 3.5.1.dev minimum mdfv4 use_display_names=True	4694	61
asammdf 3.5.1.dev full mdfv4 use_display_names=False	2020	328
asammdf 3.5.1.dev low mdfv4 use_display_names=False	1912	163
asammdf 3.5.1.dev minimum mdfv4 use_display_names=False	966	59

CHAPTER 5

Examples

5.1 Working with MDF

```
from __future__ import print_function, division
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.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]
```

(continues on next page)

(continued from previous page)

```
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 lowest 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)
```

5.2 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

# 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,
```

(continues on next page)

(continued from previous page)

```
                    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

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()
```

5.3 MF4 demo file generator

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

cycles = 100
```

(continues on next page)

(continued from previous page)

```
sigs = []

mdf = MDF()

t = np.arange(cycles, dtype=np.float64)

# no conversion
sig = Signal(
    np.ones(cycles, dtype=np.uint64),
    t,
    name='Channel_no_conversion',
    unit='s',
    conversion=None,
    comment='Unsigned 64 bit channel {}',
)
sigs.append(sig)

# linear
conversion = {
    'a': 2,
    'b': -0.5,
}
sig = Signal(
    np.ones(cycles, dtype=np.int64),
    t,
    name='Channel_linear_conversion',
    unit='Nm',
    conversion=conversion,
    comment='Signed 64bit channel with linear conversion',
)
sigs.append(sig)


# algebraic
conversion = {
    'formula': '2 * sin(X)',
}
sig = Signal(
    np.arange(cycles, dtype=np.int32) / 100.0,
    t,
    name='Channel_algebraic',
    unit='eV',
    conversion=conversion,
    comment='Sinus channel with algebraic conversion',
)
sigs.append(sig)

# rational
conversion = {
    'P1': 0,
    'P2': 4,
    'P3': -0.5,
    'P4': 0,
    'P5': 0,
    'P6': 1,
}
sig = Signal(
```

(continues on next page)

(continued from previous page)

```
    np.ones(cycles, dtype=np.int64),
    t,
    name='Channel_rational_conversion',
    unit='Nm',
    conversion=conversion,
    comment='Channel with rational conversion',
)
sigs.append(sig)

# string channel
sig = [
    'String channel sample {}'.format(j).encode('ascii')
    for j in range(cycles)
]
sig = Signal(
    np.array(sig),
    t,
    name='Channel_string',
    comment='String channel',
)
sigs.append(sig)

# byte array
ones = np.ones(cycles, dtype=np.dtype('(8,)u1'))
sig = Signal(
    ones*111,
    t,
    name='Channel_bytearay',
    comment='Byte array channel',
)
sigs.append(sig)

# tabular
vals = 20
conversion = {
    'raw_{}'.format(i): i
    for i in range(vals)
}
conversion.update(
    {
        'phys_{}'.format(i): -i
        for i in range(vals)
    }
)
sig = Signal(
    np.arange(cycles, dtype=np.uint32) % 20,
    t,
    name='Channel_tabular',
    unit='-',
    conversion=conversion,
    comment='Tabular channel',
)
sigs.append(sig)

# value to text
vals = 20
conversion = {
```

(continues on next page)

(continued from previous page)

```
    'val_{}'.format(i): i
    for i in range(vals)
}
conversion.update(
    {
        'text_{}'.format(i): 'key_{}'.format(i).encode('ascii')
        for i in range(vals)
    }
)
conversion['default'] = b'default key'
sig = Signal(
    np.arange(cycles, dtype=np.uint32) % 30,
    t,
    name='Channel_value_to_text',
    conversion=conversion,
    comment='Value to text channel',
)
sigs.append(sig)

# tabular with range
vals = 20
conversion = {
    'lower_{}'.format(i): i * 10
    for i in range(vals)
}
conversion.update(
    {
        'upper_{}'.format(i): (i + 1) * 10
        for i in range(vals)
    }
)
conversion.update(
    {
        'phys_{}'.format(i): i
        for i in range(vals)
    }
)
conversion['default'] = -1
sig = Signal(
    2 * np.arange(cycles, dtype=np.float64),
    t,
    name='Channel_value_range_to_value',
    unit='order',
    conversion=conversion,
    comment='Value range to value channel',
)
sigs.append(sig)

# value range to text
vals = 20
conversion = {
    'lower_{}'.format(i): i * 10
    for i in range(vals)
}
conversion.update(
    {
        'upper_{}'.format(i): (i + 1) * 10
```

(continues on next page)

(continued from previous page)

```
        for i in range(vals)
    }
)
conversion.update(
    {
        'text_{}'.format(i): 'Level {}'.format(i)
        for i in range(vals)
    }
)
conversion['default'] = b'Unknown level'
sig = Signal(
    6 * np.arange(cycles, dtype=np.uint64) % 240,
    t,
    name='Channel_value_range_to_text',
    conversion=conversion,
    comment='Value range to text channel',
)
sigs.append(sig)


mdf.append(sigs, 'single dimensional channels', common_timebase=True)




sigs = []

# lookup tabel with axis
samples = [
    np.ones((cycles, 2, 3), dtype=np.uint64) * 1,
    np.ones((cycles, 2), dtype=np.uint64) * 2,
    np.ones((cycles, 3), dtype=np.uint64) * 3,
]

types = [
    ('Channel_lookup_with_axis', '(2, 3)<u8'),
    ('channel_axis_1', '(2, )<u8'),
    ('channel_axis_2', '(3, )<u8'),
]

sig = Signal(
    np.core.records.fromarrays(samples, dtype=np.dtype(types)),
    t,
    name='Channel_lookup_with_axis',
    unit='A',
    comment='Array channel with axis',
)
sigs.append(sig)

# lookup tabel with default axis
samples = [
    np.ones((cycles, 2, 3), dtype=np.uint64) * 4,
]

types = [
    ('Channel_lookup_with_default_axis', '(2, 3)<u8'),
]
```

(continues on next page)

(continued from previous page)

```
sig = Signal(
    np.core.records.fromarrays(samples, dtype=np.dtype(types)),
    t,
    name='Channel_lookup_with_default_axis',
    unit='mA',
    comment='Array channel with default axis',
)
sigs.append(sig)

# structure channel composition
samples = [
    np.ones(cycles, dtype=np.uint8) * 10,
    np.ones(cycles, dtype=np.uint16) * 20,
    np.ones(cycles, dtype=np.uint32) * 30,
    np.ones(cycles, dtype=np.uint64) * 40,
    np.ones(cycles, dtype=np.int8) * -10,
    np.ones(cycles, dtype=np.int16) * -20,
    np.ones(cycles, dtype=np.int32) * -30,
    np.ones(cycles, dtype=np.int64) * -40,
]

types = [
    ('struct_channel_0', np.uint8),
    ('struct_channel_1', np.uint16),
    ('struct_channel_2', np.uint32),
    ('struct_channel_3', np.uint64),
    ('struct_channel_4', np.int8),
    ('struct_channel_5', np.int16),
    ('struct_channel_6', np.int32),
    ('struct_channel_7', np.int64),
]

sig = Signal(
    np.core.records.fromarrays(samples, dtype=np.dtype(types)),
    t,
    name='Channel_structure_composition',
    comment='Structure channel composition',
)
sigs.append(sig)


mdf.append(sigs, 'arrays', common_timebase=True)

mdf.save('demo.mf4', overwrite=True)
```

CHAPTER 6

Benchmarks

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.

6.1 Test setup

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.

asamdf 4.0.0.dev was compared against *mdfreader 2.7.8*. *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.

6.1.1 Dependencies

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

- psutil
- mdfreader

6.1.2 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)

6.2 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-10-10.0.17134-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:

- **mdf version 3.10**
  - 167 MB file size
  - 183 groups
  - 36424 channels
- **mdf version 4.00**
  - 183 MB file size
  - 183 groups
  - 36424 channels

Open file	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full mdfv3	1466	337
asammdf 4.0.0.dev low mdfv3	1372	184
asammdf 4.0.0.dev minimum mdfv3	420	70
mdfreader 2.7.8 mdfv3	2794	430
mdfreader 2.7.8 compress mdfv3	4323	129
mdfreader 2.7.8 noDataLoading mdfv3	1187	176
asammdf 4.0.0.dev full mdfv4	1786	312
asammdf 4.0.0.dev low mdfv4	1637	147
asammdf 4.0.0.dev minimum mdfv4	1099	71
mdfreader 2.7.8 mdfv4	6706	441
mdfreader 2.7.8 compress mdfv4	8542	141
mdfreader 2.7.8 noDataLoading mdfv4	4539	182

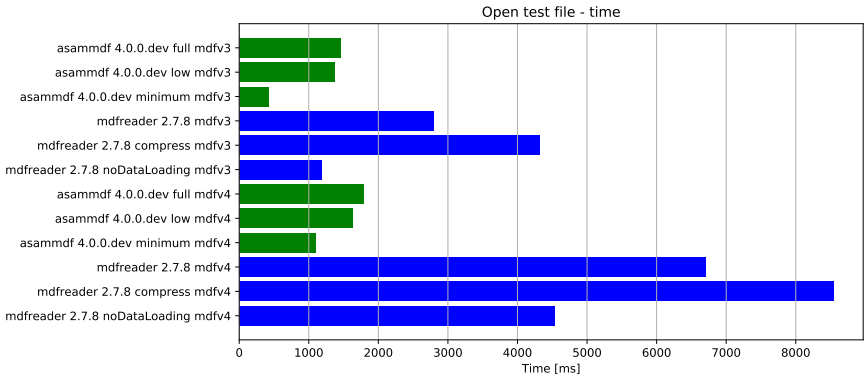
Save file	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full mdfv3	894	343
asammdf 4.0.0.dev low mdfv3	866	190
asammdf 4.0.0.dev minimum mdfv3	3135	78
mdfreader 2.7.8 mdfv3	7733	459
mdfreader 2.7.8 noDataLoading mdfv3	9353	520
mdfreader 2.7.8 compress mdfv3	7827	428
asammdf 4.0.0.dev full mdfv4	982	316
asammdf 4.0.0.dev low mdfv4	974	157
asammdf 4.0.0.dev minimum mdfv4	3600	80
mdfreader 2.7.8 mdfv4	4669	459
mdfreader 2.7.8 noDataLoading mdfv4	6612	478
mdfreader 2.7.8 compress mdfv4	4525	418

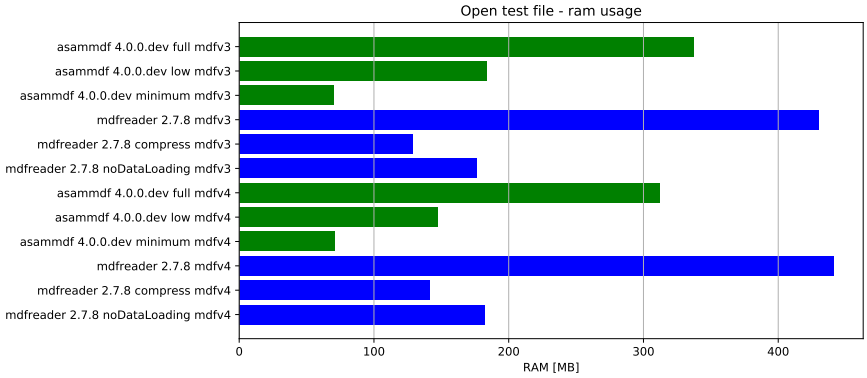
Get all channels (36424 calls)	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full mdfv3	1605	346
asammdf 4.0.0.dev low mdfv3	7224	199
asammdf 4.0.0.dev minimum mdfv3	9965	87
mdfreader 2.7.8 mdfv3	102	430
mdfreader 2.7.8 nodata mdfv3	16696	211
mdfreader 2.7.8 compress mdfv3	622	129
asammdf 4.0.0.dev full mdfv4	1685	316
asammdf 4.0.0.dev low mdfv4	12592	157
asammdf 4.0.0.dev minimum mdfv4	16428	84
mdfreader 2.7.8 mdfv4	93	441
mdfreader 2.7.8 compress mdfv4	624	141
mdfreader 2.7.8 nodata mdfv4	27146	206

Convert file	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full v3 to v4	5677	680
asammdf 4.0.0.dev low v3 to v4	5737	352
asammdf 4.0.0.dev minimum v3 to v4	9341	118
asammdf 4.0.0.dev full v4 to v3	5095	610
asammdf 4.0.0.dev low v4 to v3	5328	263
asammdf 4.0.0.dev minimum v4 to v3	9983	115

Merge 3 files	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full v3	17059	1641
asammdf 4.0.0.dev low v3	16730	622
asammdf 4.0.0.dev minimum v3	25156	166
mdfreader 2.7.8 v3	24608	1335
mdfreader 2.7.8 compress v3	30669	1347
mdfreader 2.7.8 nodata v3	24093	1456
asammdf 4.0.0.dev full v4	17949	1513
asammdf 4.0.0.dev low v4	17592	461
asammdf 4.0.0.dev minimum v4	36417	166
mdfreader 2.7.8 v4	36287	1326
mdfreader 2.7.8 nodata v4	35904	1361
mdfreader 2.7.8 compress v4	42410	1336

6.2.1 Graphical results





6.3 x86 Python results

Benchmark environment

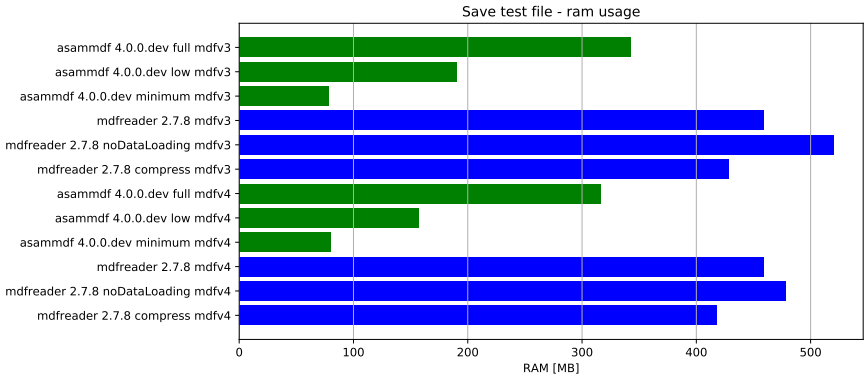
- 3.6.3 (v3.6.3:2c5fed8, Oct 3 2017, 17:26:49) [MSC v.1900 32 bit (Intel)]
- Windows-10-10.0.17134-SP0
- Intel64 Family 6 Model 69 Stepping 1, GenuineIntel
- 16GB installed RAM

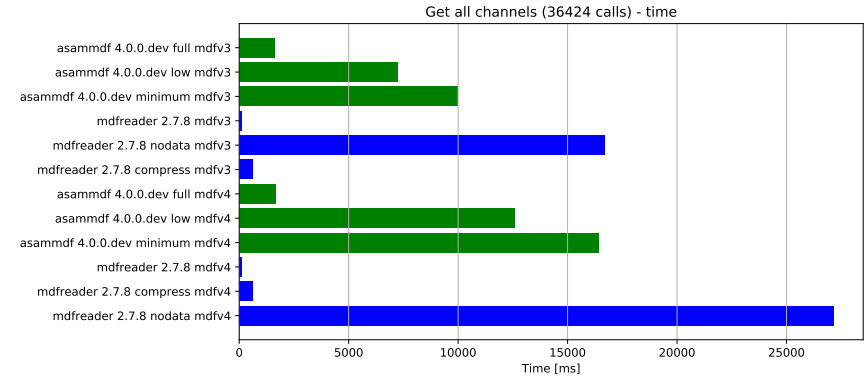
Notations used in the results

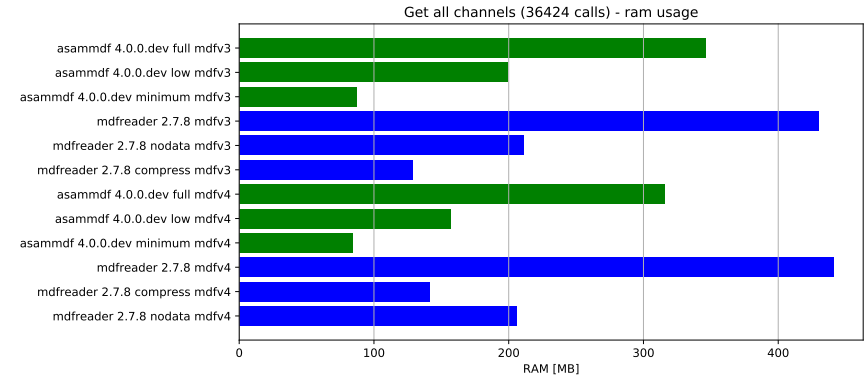
- full = asammdf MDF object created with memory=full (everything loaded into RAM)
- low = asammdf MDF object created with memory=low (raw channel data not loaded into RAM, but metadata loaded to RAM)
- minimum = asammdf MDF object created with memory=full (lowest possible RAM usage)
- compress = mdfreader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

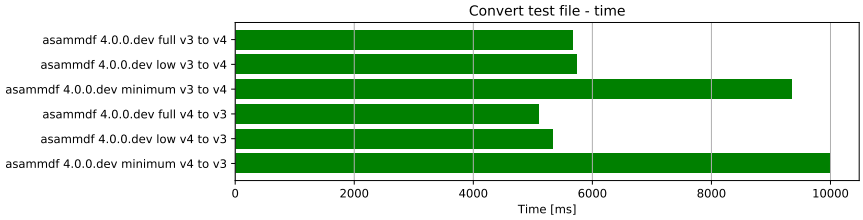
Files used for benchmark:

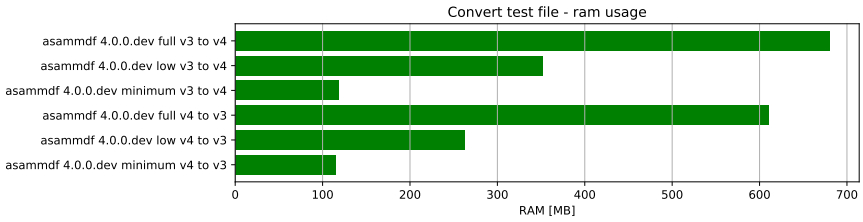

Save test file - time
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 noDataLoading mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 noDataLoading mdfv4
mdfreader 2.7.8 compress mdfv4
0
2000
4000
6000
8000
Time [ms]

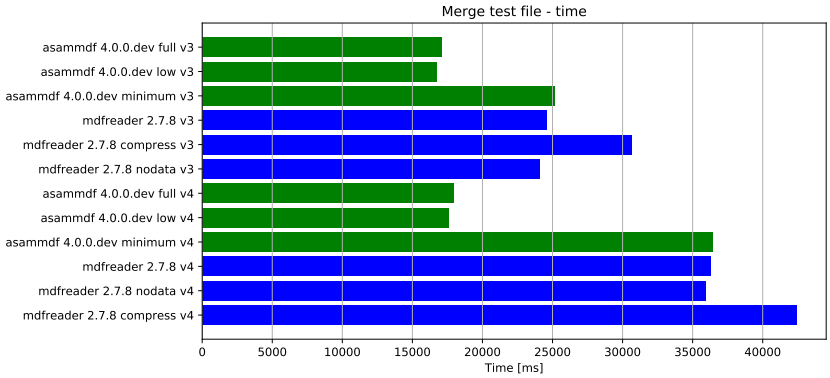

Save test file - ram usage
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 noDataLoading mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 noDataLoading mdfv4
mdfreader 2.7.8 compress mdfv4
0
100
200
300
400
500
RAM [MB]

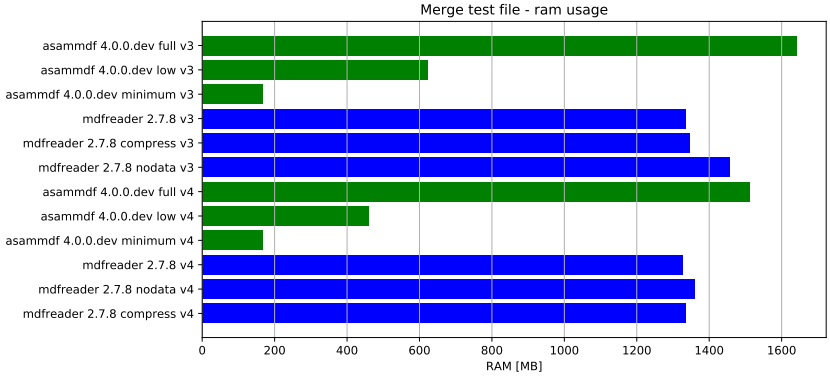

Get all channels (36424 calls) - time
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 nodata mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 compress mdfv4
mdfreader 2.7.8 nodata mdfv4
0
5000
10000
15000
20000
25000
Time [ms]


Get all channels (36424 calls) - ram usage
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 nodata mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 compress mdfv4
mdfreader 2.7.8 nodata mdfv4
0
100
200
300
400
RAM [MB]


Convert test file - time
asammdf 4.0.0.dev full v3 to v4
asammdf 4.0.0.dev low v3 to v4
asammdf 4.0.0.dev minimum v3 to v4
asammdf 4.0.0.dev full v4 to v3
asammdf 4.0.0.dev low v4 to v3
asammdf 4.0.0.dev minimum v4 to v3
0
2000
4000
6000
8000
10000
Time [ms]


Convert test file - ram usage
asammdf 4.0.0.dev full v3 to v4
asammdf 4.0.0.dev low v3 to v4
asammdf 4.0.0.dev minimum v3 to v4
asammdf 4.0.0.dev full v4 to v3
asammdf 4.0.0.dev low v4 to v3
asammdf 4.0.0.dev minimum v4 to v3
0
100
200
300
400
500
600
700
RAM [MB]


Merge test file - time
asammdf 4.0.0.dev full v3
asammdf 4.0.0.dev low v3
asammdf 4.0.0.dev minimum v3
mdfreader 2.7.8 v3
mdfreader 2.7.8 compress v3
mdfreader 2.7.8 nodata v3
asammdf 4.0.0.dev full v4
asammdf 4.0.0.dev low v4
asammdf 4.0.0.dev minimum v4
mdfreader 2.7.8 v4
mdfreader 2.7.8 nodata v4
mdfreader 2.7.8 compress v4
0
5000
10000
15000
20000
25000
30000
35000
40000
Time [ms]


Merge test file - ram usage
asammdf 4.0.0.dev full v3
asammdf 4.0.0.dev low v3
asammdf 4.0.0.dev minimum v3
mdfreader 2.7.8 v3
mdfreader 2.7.8 compress v3
mdfreader 2.7.8 nodata v3
asammdf 4.0.0.dev full v4
asammdf 4.0.0.dev low v4
asammdf 4.0.0.dev minimum v4
mdfreader 2.7.8 v4
mdfreader 2.7.8 nodata v4
mdfreader 2.7.8 compress v4
0
200
400
600
800
1000
1200
1400
1600
RAM [MB]

- **mdf version 3.10**
  - 167 MB file size
  - 183 groups
  - 36424 channels
- **mdf version 4.00**
  - 183 MB file size
  - 183 groups
  - 36424 channels

Open file	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full mdfv3	1792	272
asammdf 4.0.0.dev low mdfv3	1650	119
asammdf 4.0.0.dev minimum mdfv3	556	49
mdfreader 2.7.8 mdfv3	3526	394
mdfreader 2.7.8 compress mdfv3	5373	96
mdfreader 2.7.8 noDataLoading mdfv3	1427	108
asammdf 4.0.0.dev full mdfv4	2181	259
asammdf 4.0.0.dev low mdfv4	2045	94
asammdf 4.0.0.dev minimum mdfv4	1396	49
mdfreader 2.7.8 mdfv4	8189	399
mdfreader 2.7.8 compress mdfv4	10217	100
mdfreader 2.7.8 noDataLoading mdfv4	5227	110

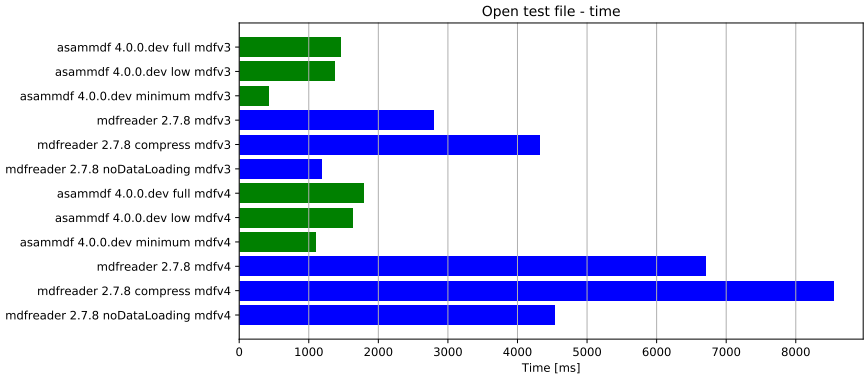
Save file	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full mdfv3	1150	278
asammdf 4.0.0.dev low mdfv3	1303	125
asammdf 4.0.0.dev minimum mdfv3	3978	54
mdfreader 2.7.8 mdfv3	9154	414
mdfreader 2.7.8 noDataLoading mdfv3	10710	450
mdfreader 2.7.8 compress mdfv3	8789	388
asammdf 4.0.0.dev full mdfv4	1262	262
asammdf 4.0.0.dev low mdfv4	1382	103
asammdf 4.0.0.dev minimum mdfv4	4015	58
mdfreader 2.7.8 mdfv4	5285	418
mdfreader 2.7.8 noDataLoading mdfv4	8016	437
mdfreader 2.7.8 compress mdfv4	5027	382

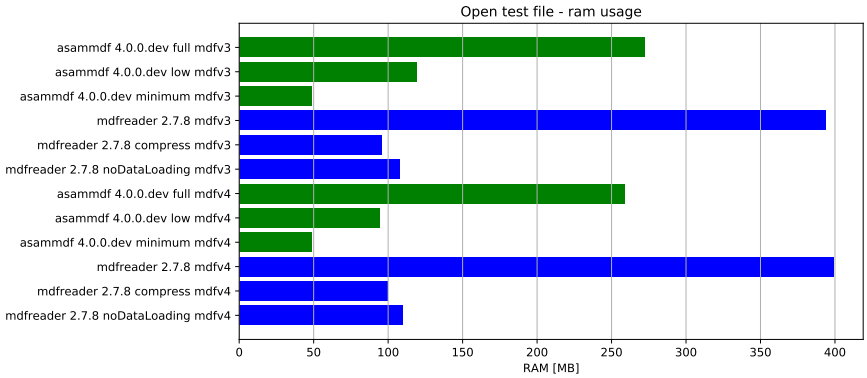
Get all channels (36424 calls)	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full mdfv3	2052	280
asammdf 4.0.0.dev low mdfv3	21601	131
asammdf 4.0.0.dev minimum mdfv3	24989	63
mdfreader 2.7.8 mdfv3	125	394
mdfreader 2.7.8 nodata mdfv3	33777	132
mdfreader 2.7.8 compress mdfv3	676	96
asammdf 4.0.0.dev full mdfv4	2177	263
asammdf 4.0.0.dev low mdfv4	25377	103
asammdf 4.0.0.dev minimum mdfv4	29231	58
mdfreader 2.7.8 mdfv4	116	399
mdfreader 2.7.8 compress mdfv4	666	105
mdfreader 2.7.8 nodata mdfv4	42202	125

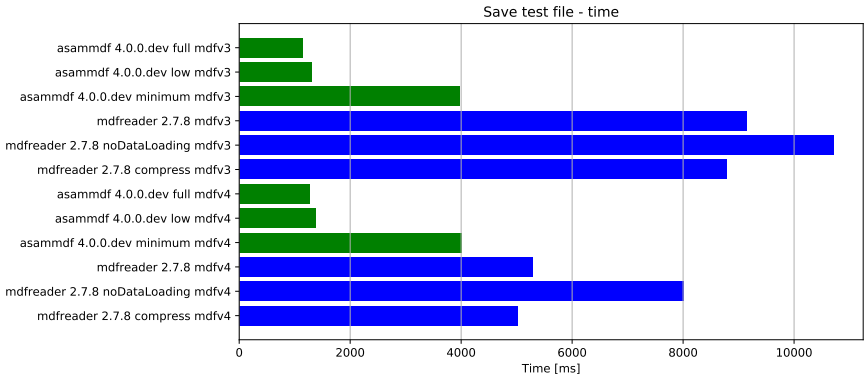
Convert file	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full v3 to v4	6679	552
asammdf 4.0.0.dev low v3 to v4	6897	221
asammdf 4.0.0.dev minimum v3 to v4	11120	87
asammdf 4.0.0.dev full v4 to v3	6004	511
asammdf 4.0.0.dev low v4 to v3	6294	161
asammdf 4.0.0.dev minimum v4 to v3	12459	79

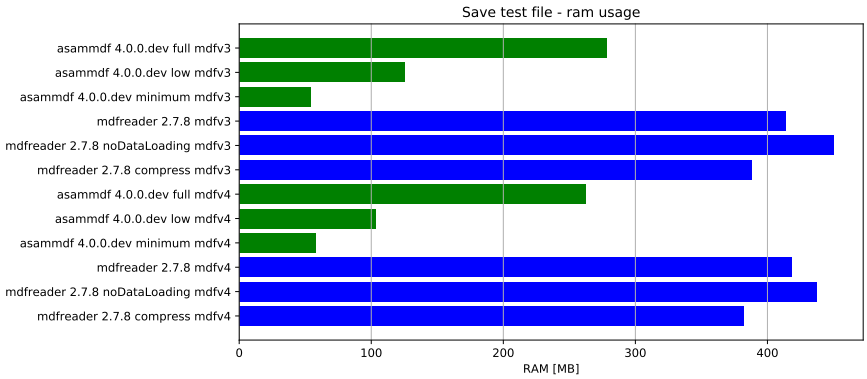
Merge 3 files	Time [ms]	RAM [MB]
asammdf 4.0.0.dev full v3	20648	1411
asammdf 4.0.0.dev low v3	20279	391
asammdf 4.0.0.dev minimum v3	30807	119
mdfreader 2.7.8 v3	27054	1274
mdfreader 2.7.8 compress v3	32342	1288
mdfreader 2.7.8 nodata v3	26471	1343
asammdf 4.0.0.dev full v4	21532	1335
asammdf 4.0.0.dev low v4	21272	280
asammdf 4.0.0.dev minimum v4	45546	111
mdfreader 2.7.8 v4	40785	1260
mdfreader 2.7.8 nodata v4	40467	1284

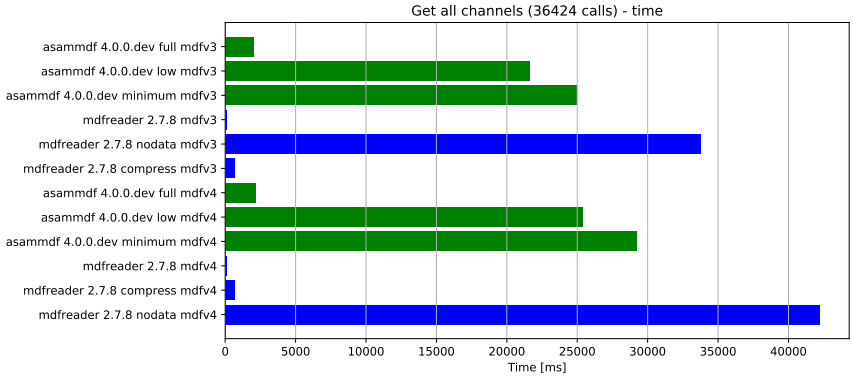
6.3.1 Graphical results

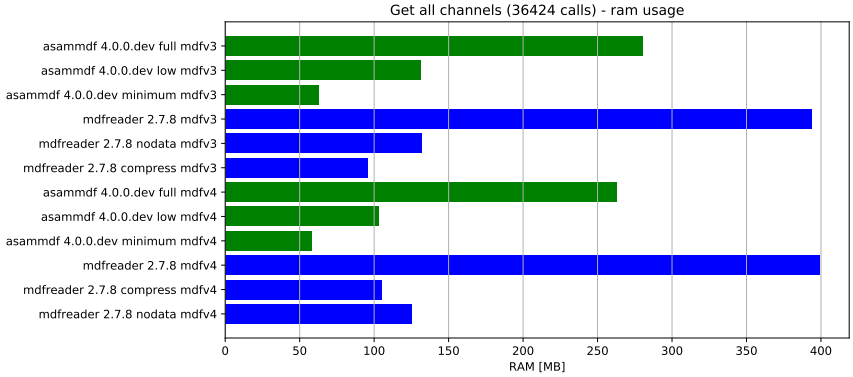

Open test file - time
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 compress mdfv3
mdfreader 2.7.8 noDataLoading mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 compress mdfv4
mdfreader 2.7.8 noDataLoading mdfv4
0
1000
2000
3000
4000
5000
6000
7000
8000
Time [ms]

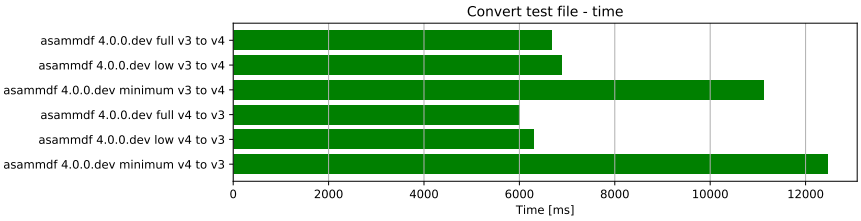

Open test file - ram usage
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 compress mdfv3
mdfreader 2.7.8 noDataLoading mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 compress mdfv4
mdfreader 2.7.8 noDataLoading mdfv4
0
50
100
150
200
250
300
350
400
RAM [MB]

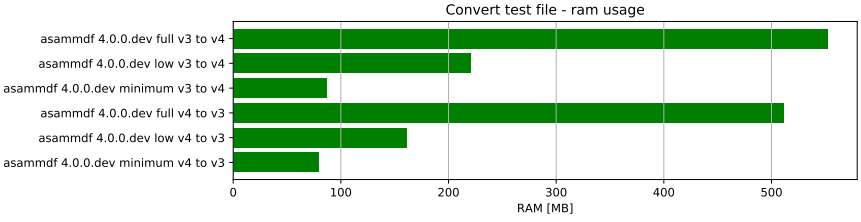

Save test file - time
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 noDataLoading mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 noDataLoading mdfv4
mdfreader 2.7.8 compress mdfv4
0
2000
4000
6000
8000
10000
Time [ms]

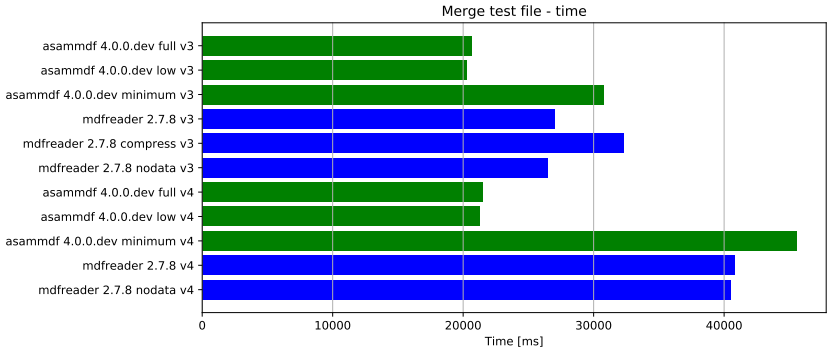

Save test file - ram usage
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 noDataLoading mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 noDataLoading mdfv4
mdfreader 2.7.8 compress mdfv4
0
100
200
300
400
RAM [MB]

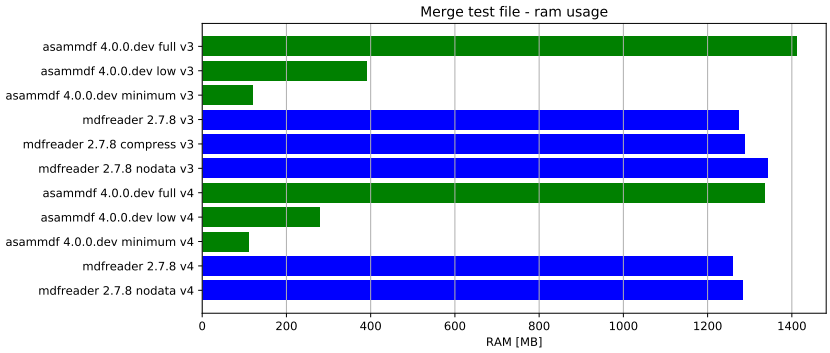

Get all channels (36424 calls) - time
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 nodata mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 compress mdfv4
mdfreader 2.7.8 nodata mdfv4
0
5000
10000
15000
20000
25000
30000
35000
40000
Time [ms]


Get all channels (36424 calls) - ram usage
asammdf 4.0.0.dev full mdfv3
asammdf 4.0.0.dev low mdfv3
asammdf 4.0.0.dev minimum mdfv3
mdfreader 2.7.8 mdfv3
mdfreader 2.7.8 nodata mdfv3
mdfreader 2.7.8 compress mdfv3
asammdf 4.0.0.dev full mdfv4
asammdf 4.0.0.dev low mdfv4
asammdf 4.0.0.dev minimum mdfv4
mdfreader 2.7.8 mdfv4
mdfreader 2.7.8 compress mdfv4
mdfreader 2.7.8 nodata mdfv4
0
50
100
150
200
250
300
350
400
RAM [MB]


Convert test file - time
asammdf 4.0.0.dev full v3 to v4
asammdf 4.0.0.dev low v3 to v4
asammdf 4.0.0.dev minimum v3 to v4
asammdf 4.0.0.dev full v4 to v3
asammdf 4.0.0.dev low v4 to v3
asammdf 4.0.0.dev minimum v4 to v3
0
2000
4000
6000
8000
10000
12000
Time [ms]


Convert test file - ram usage
asammdf 4.0.0.dev full v3 to v4
asammdf 4.0.0.dev low v3 to v4
asammdf 4.0.0.dev minimum v3 to v4
asammdf 4.0.0.dev full v4 to v3
asammdf 4.0.0.dev low v4 to v3
asammdf 4.0.0.dev minimum v4 to v3
0
100
200
300
400
500
RAM [MB]


Merge test file - time
asammdf 4.0.0.dev full v3
asammdf 4.0.0.dev low v3
asammdf 4.0.0.dev minimum v3
mdfreader 2.7.8 v3
mdfreader 2.7.8 compress v3
mdfreader 2.7.8 nodata v3
asammdf 4.0.0.dev full v4
asammdf 4.0.0.dev low v4
asammdf 4.0.0.dev minimum v4
mdfreader 2.7.8 v4
mdfreader 2.7.8 nodata v4
0
10000
20000
30000
40000
Time [ms]


Merge test file - ram usage
asammdf 4.0.0.dev full v3
asammdf 4.0.0.dev low v3
asammdf 4.0.0.dev minimum v3
mdfreader 2.7.8 v3
mdfreader 2.7.8 compress v3
mdfreader 2.7.8 nodata v3
asammdf 4.0.0.dev full v4
asammdf 4.0.0.dev low v4
asammdf 4.0.0.dev minimum v4
mdfreader 2.7.8 v4
mdfreader 2.7.8 nodata v4
0
200
400
600
800
1000
1200
1400
RAM [MB]

CHAPTER 7

GUI

Starting with version 3.4.0 there is a graphical user interface that comes together with *asammdf*.

With the GUI tool you can

- visualize channels
- see channel, conversion and source metadata as stored in the MDF file
- access library functionality for single files (convert, export, cut, filter, resample) and multiple files (concatenate, stack)

After you pip install asammdf there will be a new script called *asammdf.exe* in the ``python_installation_folder\Scripts`` folder.

The following dependencies are required by the GUI

- PyQt4 or PyQt5
- pyqtgraph

7.1 Menu

7.1.1 File

The only command here is *Open*. Depending on the page this is allow to open a single file, or multiple files.

7.1.2 Settings

The following settings are available

- **Memory**: this specifies the *memory* argument when instantiating the *MDF* object. It can have one of the values:
  - `full`
  - `low`

  - `minimum`

  Changing this option does not affect already opened files; it will only apply later when openeing a new file.

- **Plot lines**: controls the visual style of the channels plot lines:
  - `Simple`: a simple line is used to join the channel samples. It is the default option because it gives the best performance for high sample count
  - `With dots`: a simple line joins the channels sample, and the actual samples are marked by a dot. This allows for better visualization, but at the expense of lower performance
- **Integer line style**: controls how integer type channels are displayed
  - `Step mode`: the line that joins the channel samples is a step line
  - `Direct connect mode`: the samples are joined using straight lines
- **Search**: controls how the matching is done for the quick search field. Mathing is always done case insensitive.
  - `Match start`: the channel name must start with the input string
  - `Match contains`: the channel name must contain the input string

Note:

- Changing the *Memory* option does not affect already opened files; it will only apply later when openeing a new file.

7.1.3 Plot

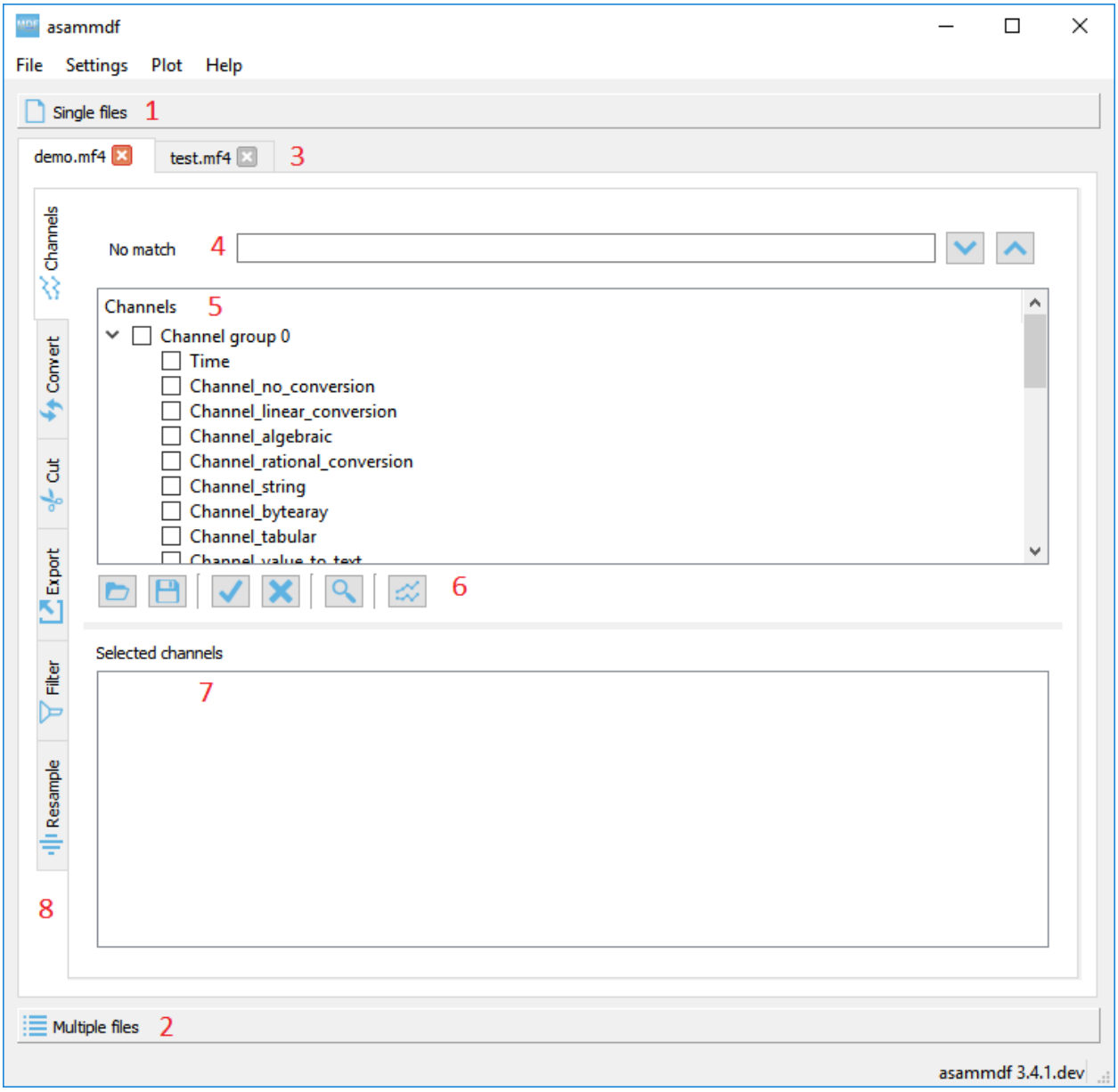
There are several keyboard shortcuts for handling the plots:

Short-cut	Action	Desctiption
C	Cursor	Displays a movable cursor that will trigger the display of the current value for all plot channels
F	Fit	Y-axis fit all active channels on the screen, keeping the current X-axis range
G	Grid	Toggle grid lines
H	Home	XY-axis fit all active channels
I	Zoom-in	X-axis zoom-in[1]
O	Zoom-out	X-axis zoom-out[1]
M	Statistics	Toggle the display of the statistic panel
R	Range	Display a movable range that will trigger the display of the delta values for all plot channels
S	Stack	Y Stack all active channels so that they don't overlap, keeping the X-axis range
←	Move cursor left	Moves the cursor to the next sample on the left
→	Move cursor right	Moves the cursor to the next sample on the right
Ctrl+B	Bin	Toggle binary reprezentation of integer channels
Ctrl+H	Hex	Toggle hex reprezentation of integer channels
Ctrl+P	Physical	Toggle physical reprezentation of integer channels

[1] If the cursor is prezent the zooming will center on it.

7.2 Single files

The *Single files* toolbox page is used to open multiple single files for visualization and processing (for example exporting to csv or hdf5).



1. Single files page selector
2. Multiple files page selector
3. Opened files tabs
4. Quick channel search field for the current file
5. Complete channels tree
6. Command buttons
7. Selected channels list

8. Current file operations tabs

7.2.1 Opened files tabs

In the single files mode, you can open multiple files in parallel. The tab names have the title set to the short file name, and the complete file path can be seen as the tab tooltip.

There is no restriction, so the same file can be opened several times.

7.2.2 Quick channel search field for the current file

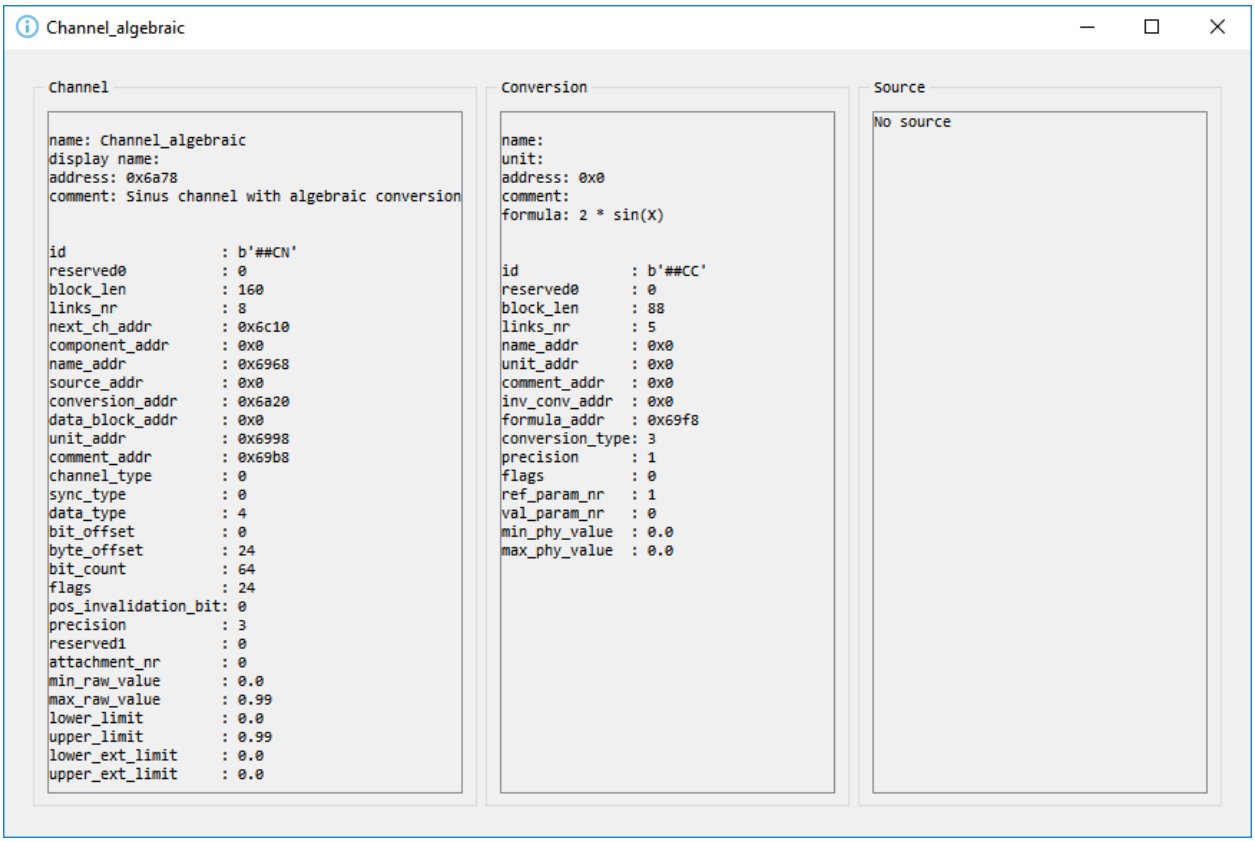
Using the *Seetings->Searh* menu option the user can choose how the search is performed. A positive search match will scroll the channels tree and highlight the channel entry.

When the same channel name exist several times in the file, you can switch between the occurences using the arrow buttons.

7.2.3 Complete channels tree

This tree contains all the channels found in the measurement.

Double clicking a channel name will display a pop-up window with the channel information (CNBLOCK, CCBLOCK and SIBLOCK/CEBLOCK)

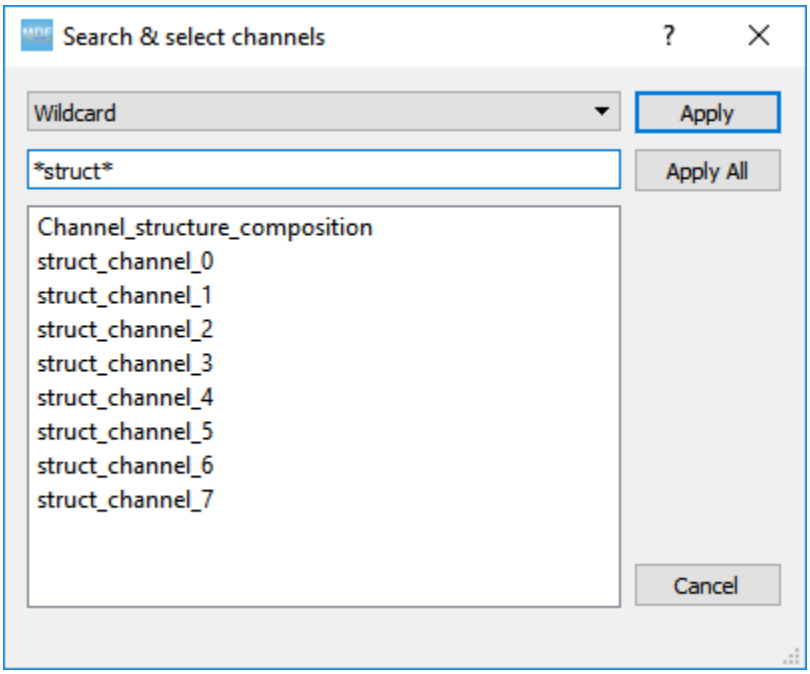


Only the channels that are checked in the channels tree will pe selected for plotting when the *Plot* button willbe pressed. Checking or unchecking channels will not affect the current plot.

7.2.4 Command buttons

From left to right the buttons have the following functionality

- **Load channel selection list**: loads a channel selection list from a text file (one channel name per line) and checks them in the channels tree is they are found.
- **Save channel selection list**: saves the current checked channels names in a text file
- **Select all channels**: checks all channels in the channels tree
- **Reset selection**: unchecks all channels in the channels tree
- **Advanced search & select**: will open an advanced search dialog
  - the dialog can use wildcard and regex patterns
  - multiple channels can be selected, and thus checked in the channels tree



- **Plot**: generates the plot based on the current checked channels from the channels tree

7.2.5 Selected channels list

When the *Plot* button is pressed the checked channels will populate the *Selected channels list*.

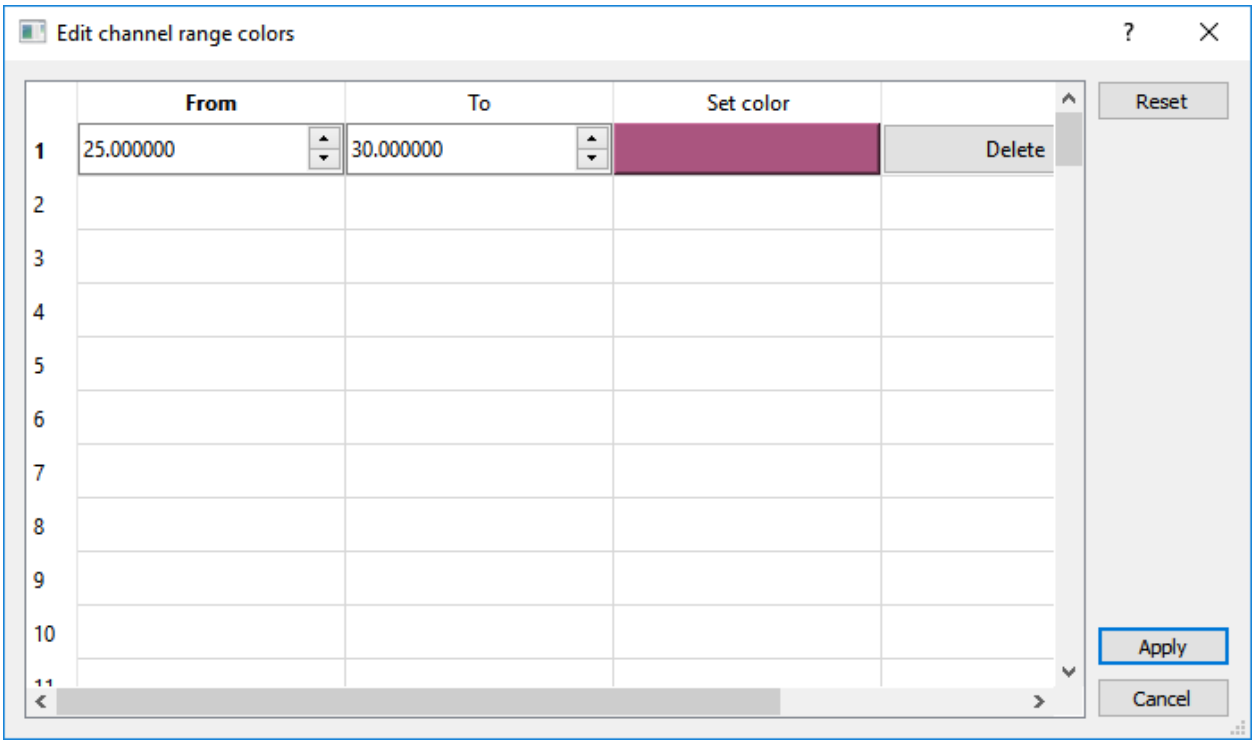
Selecting items from the *Selected channels list* will display their Y-axis on the right side of the plot, if the items are enabled for display.

It also necessary to select a single item when the *Statistics* panel is active to compute the statistics for the item's channel.

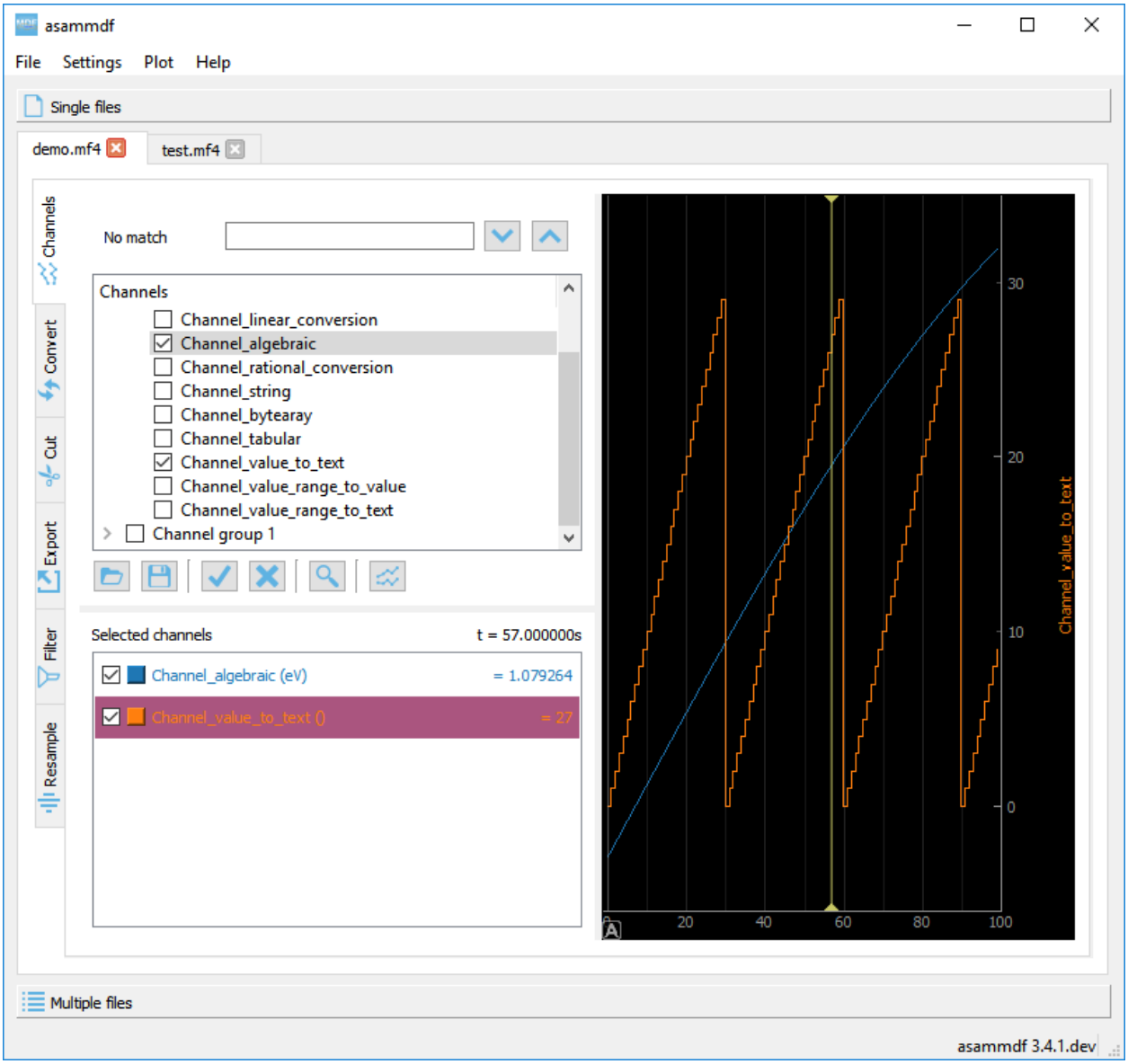
Each item has four elements

1. display enable checkbox
2. color select button
3. channel name and unit label
4. channel value label
    - the value is only displayed if the cursor or range are active. For the cursor is will show the current value, and for the range it will shwo the value delta between the range start and stop timestamps

Double clicking an item will open a range editor dialog

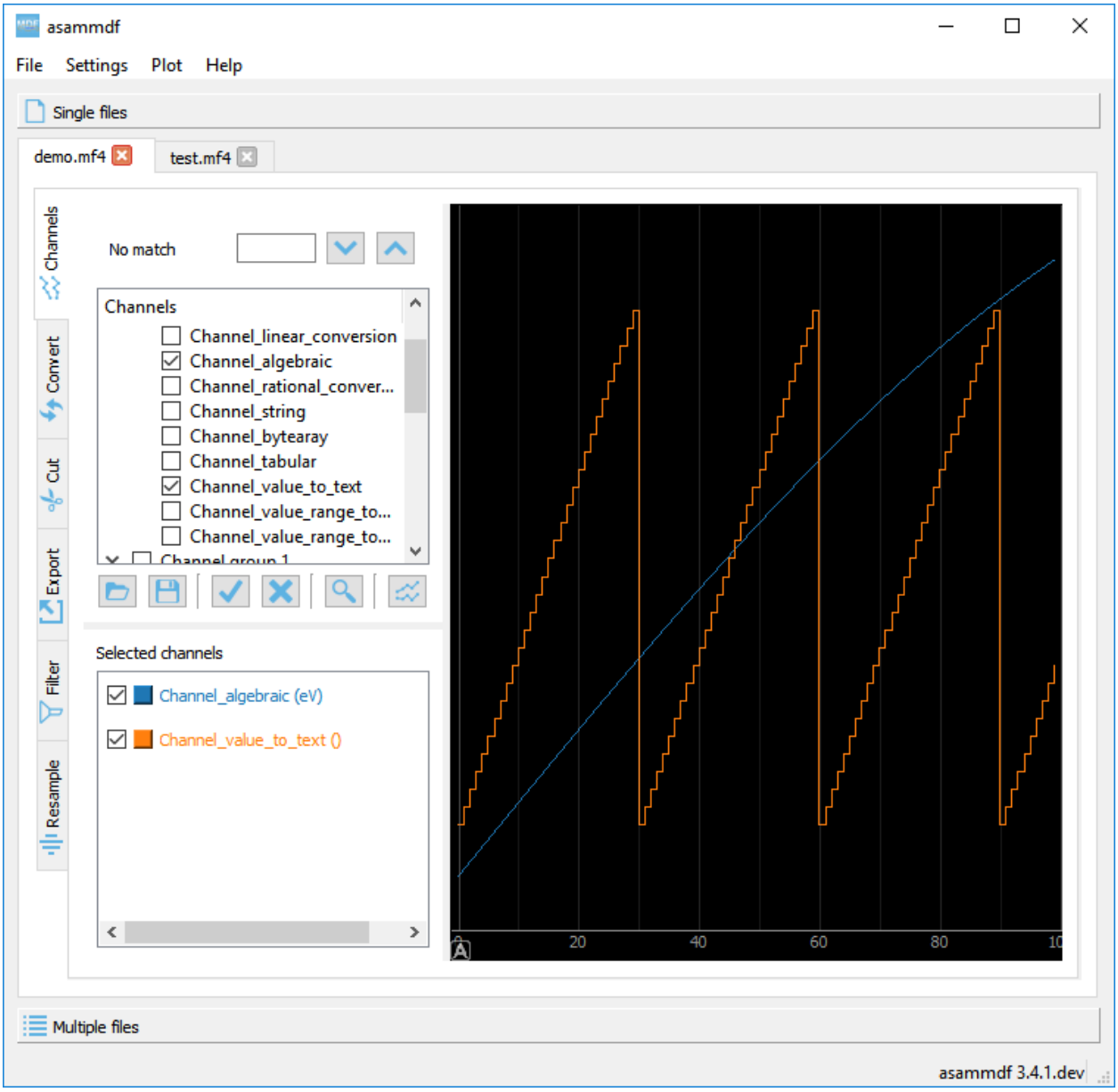


Here we can specify a range value visual alert. When the cursor is active and the current channel value is within the specified range, the item background will change to the selected color.

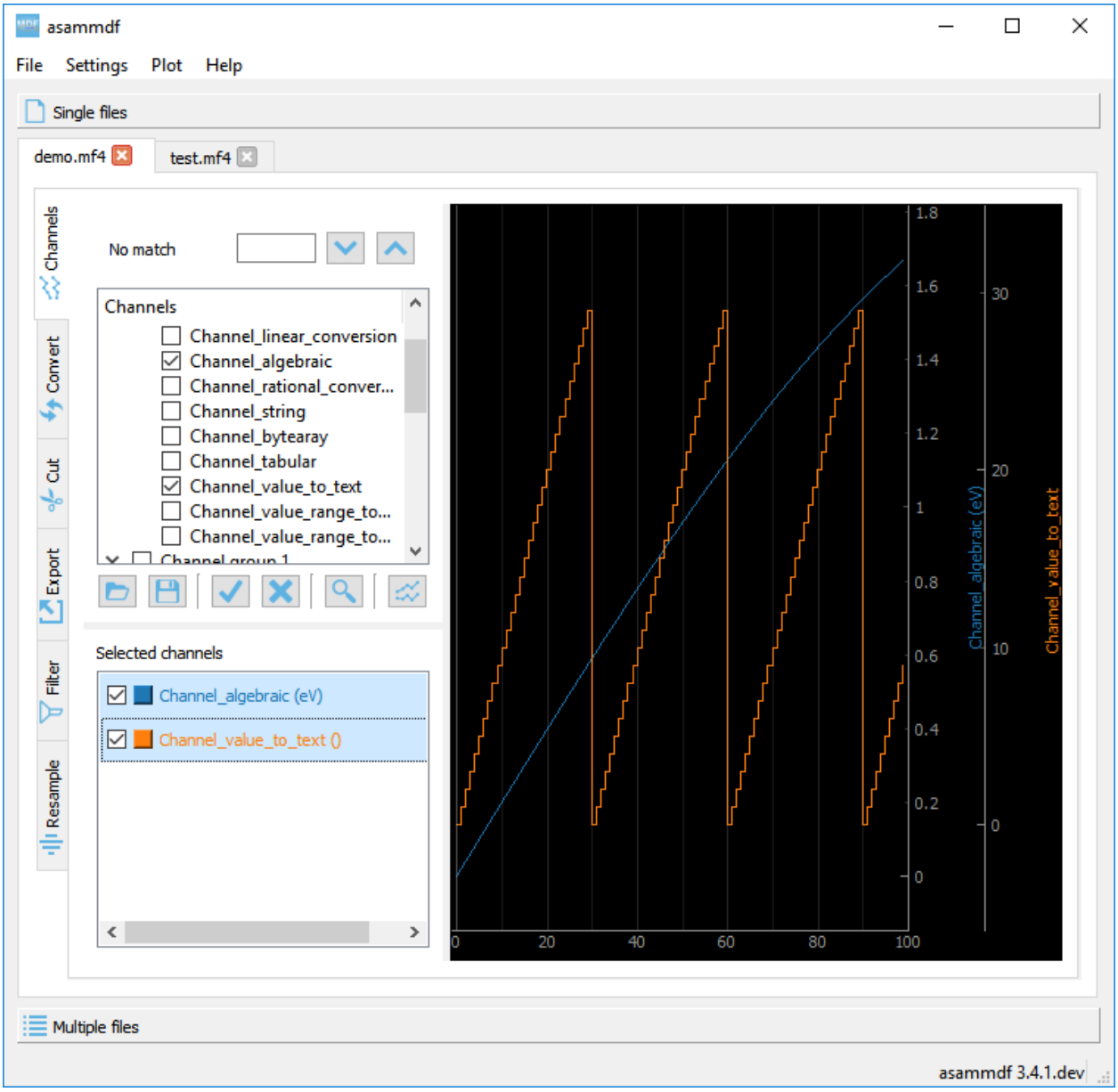


7.2.6 Plot

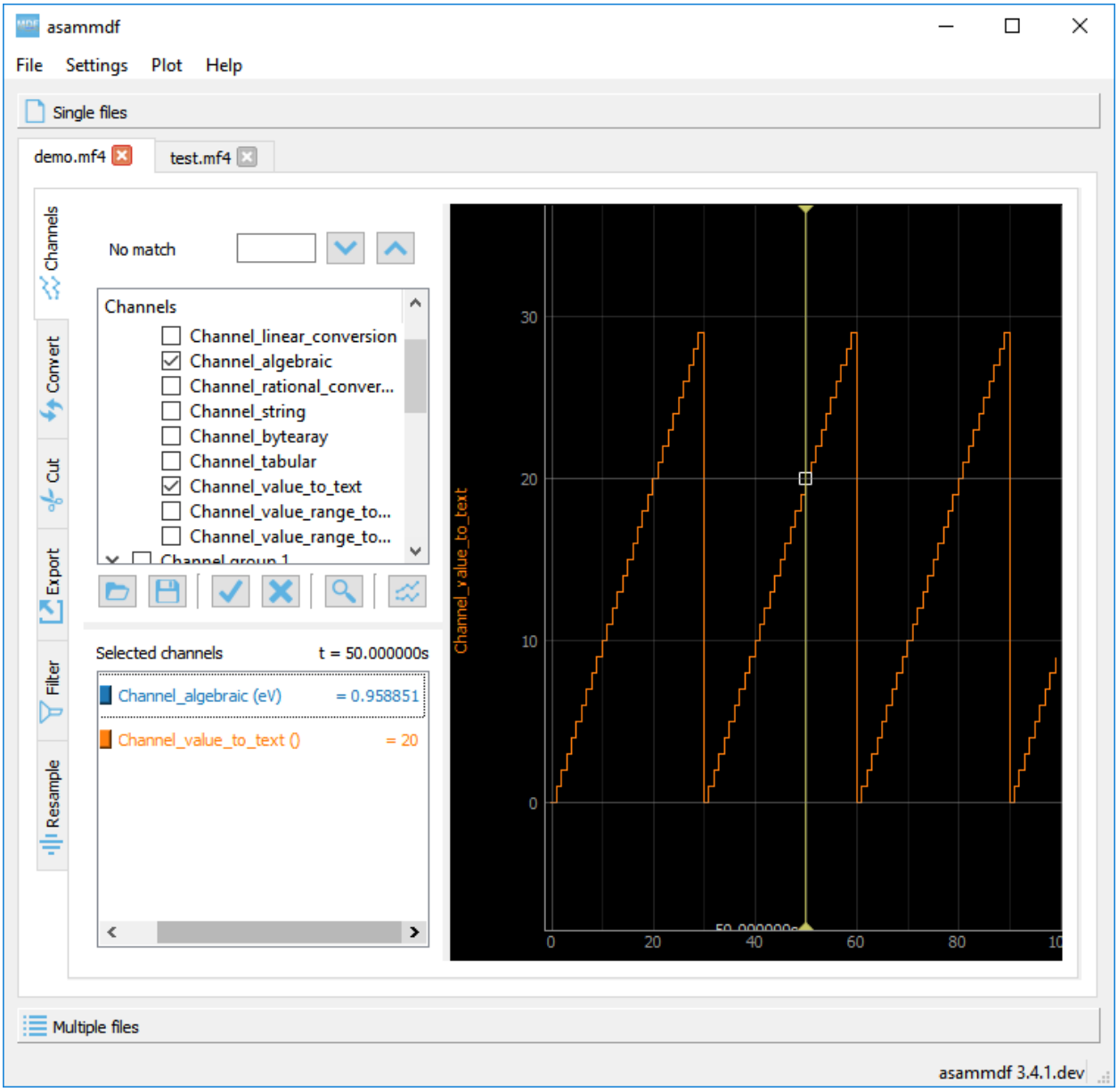
The initial plot will have all channels homed (see the *H* keyboard shortcut) and Y-axis disabled



Selecting items from the *Selected channels list* will enable the Y-axis

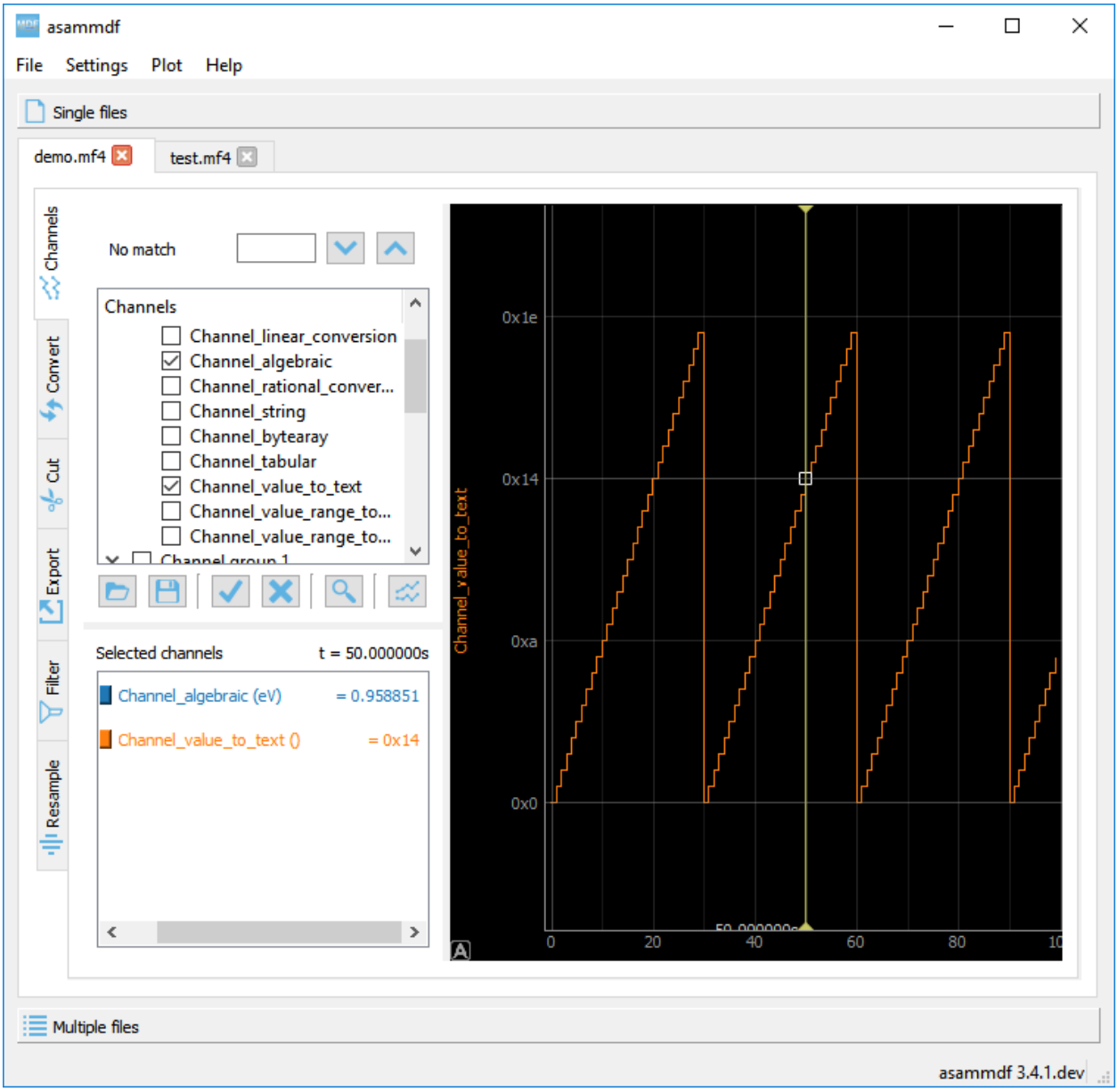


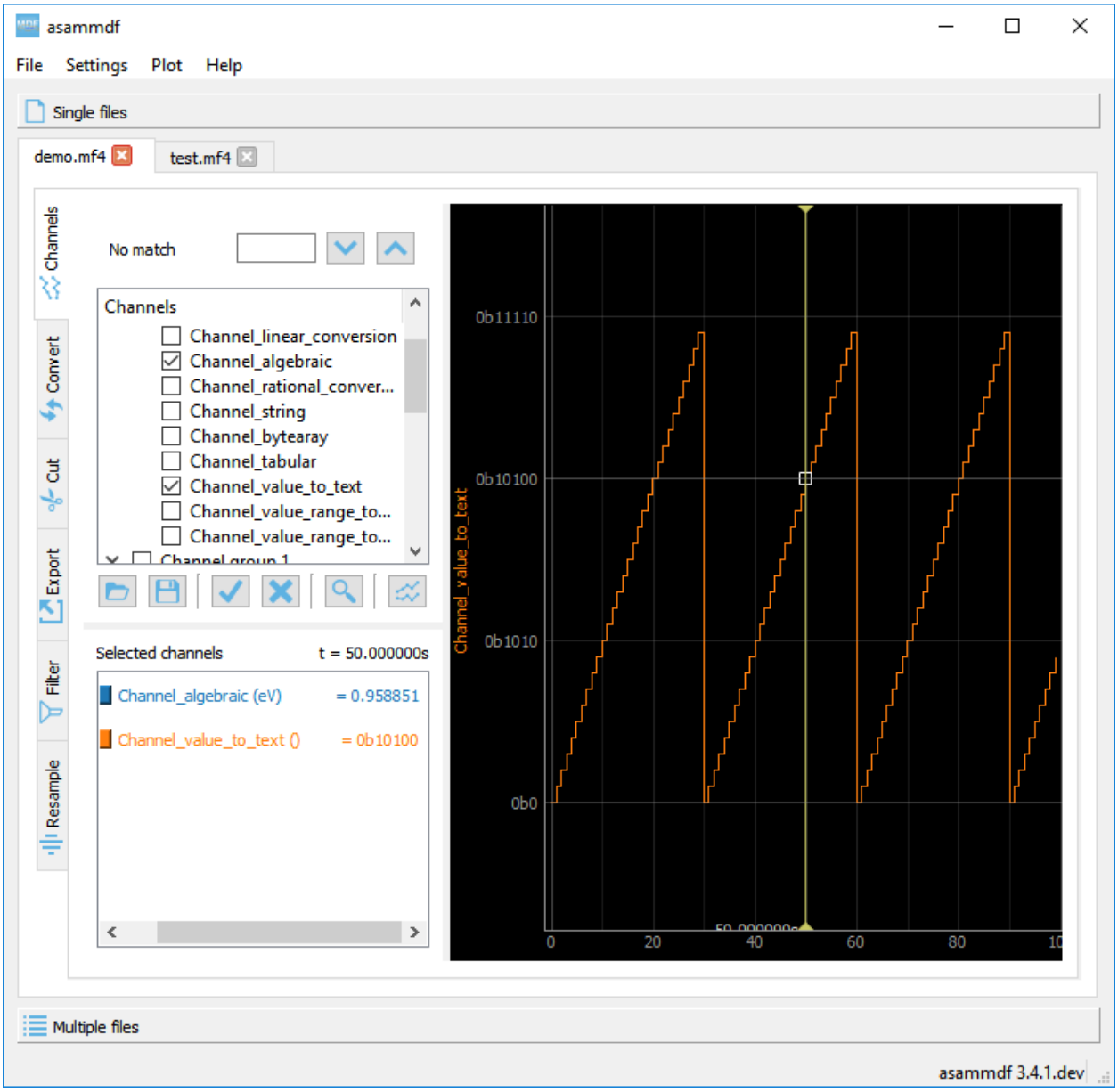
Using the *C* keyboard shotcut will toggle the cursor, and with it the channel values will be displayed for each item in the *Selected channels list*



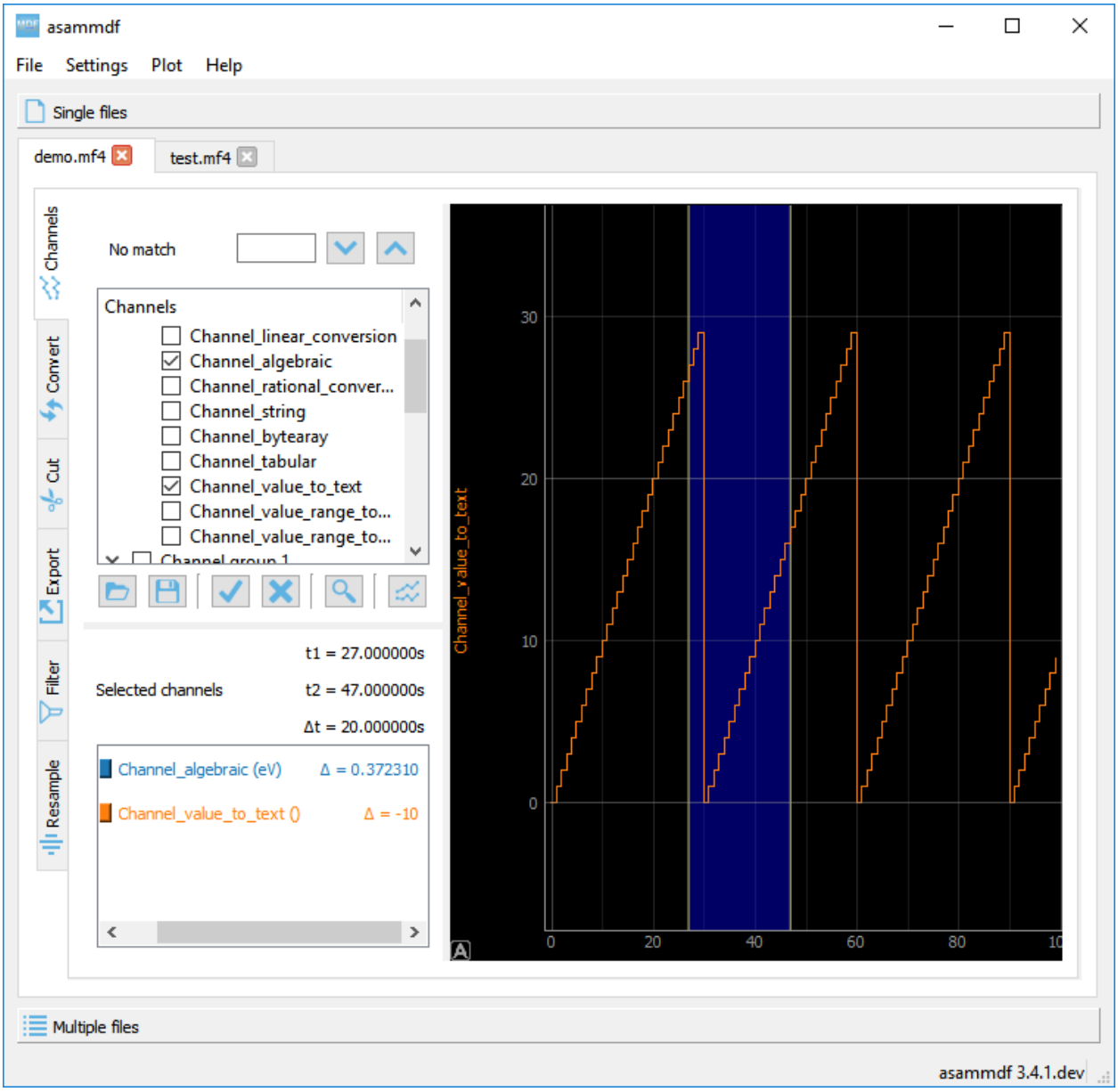
The *Ctrl+H* and *Ctrl+B* keyboard shortcuts will

- change the axis values for integer channels to hex and bin mode
- change the channel value display mode for each integer channel item in the *Selected channels list*

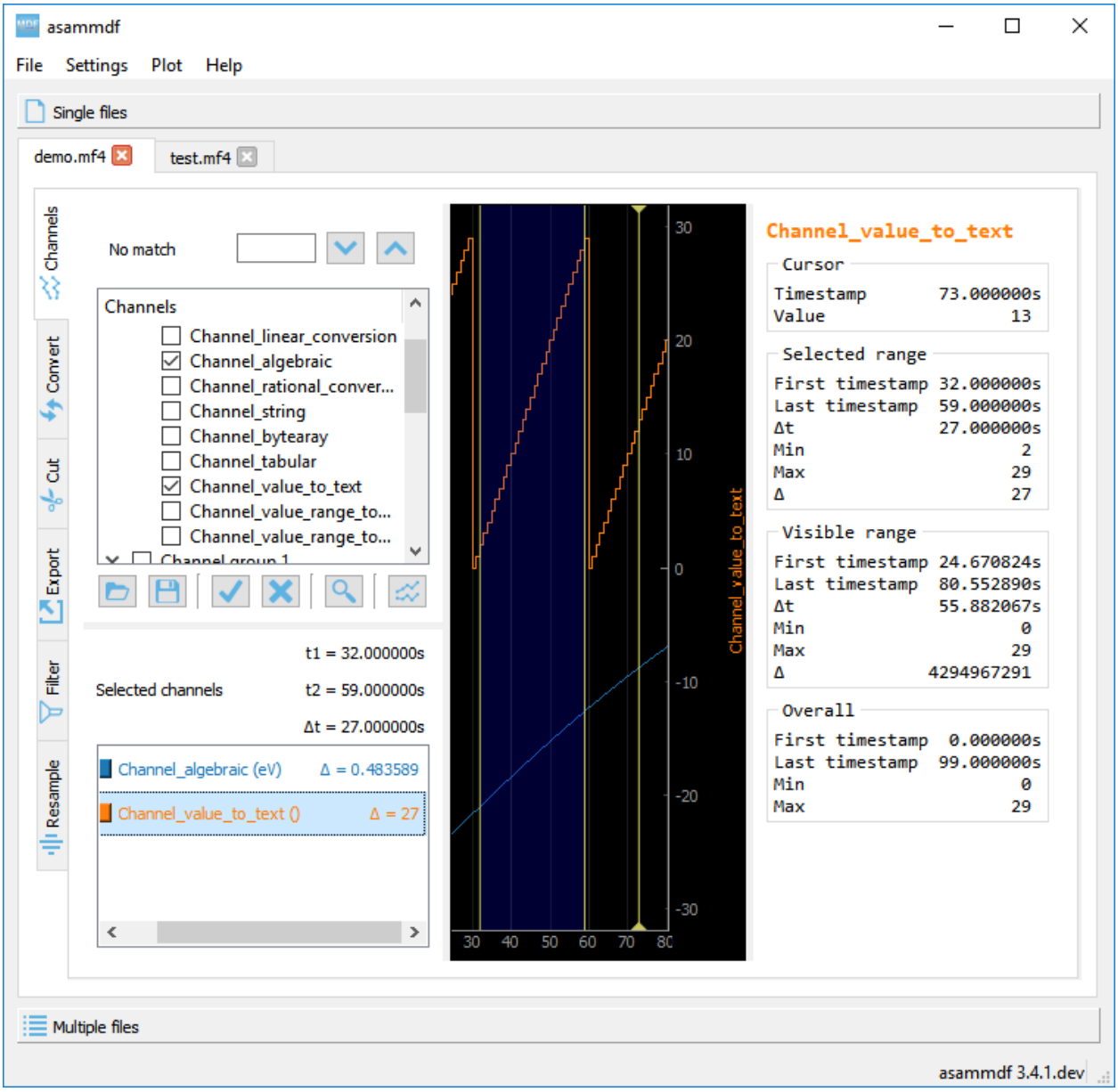

asammdf
File
Settings
Plot
Help
Single files
demo.mf4
test.mf4
Channels
Convert
Cut
Export
Filter
Resample
No match
Channels
Channel_linear_conversion
Channel_algebraic
Channel_rational_conver...
Channel_string
Channel_bytearay
Channel_tabular
Channel_value_to_text
Channel_value_range_to...
Channel_value_range_to...
Selected channels
t = 50.000000s
Channel_algebraic (eV)
= 0.958851
Channel_value_to_text ()
= 0x14
0x1e
0x14
0xa
0x0
Channel_value_to_text
0
20
40
60
80
Multiple files
asammdf 3.4.1.dev



Using the *R* keyboard shotcut will toggle the range, and with it the channel values will be displayed for each item in the *Selected channels list*. When the range is enabled, using the *H* keyboard shortcut will not home to the whole time range, but instead will use the range time interval.

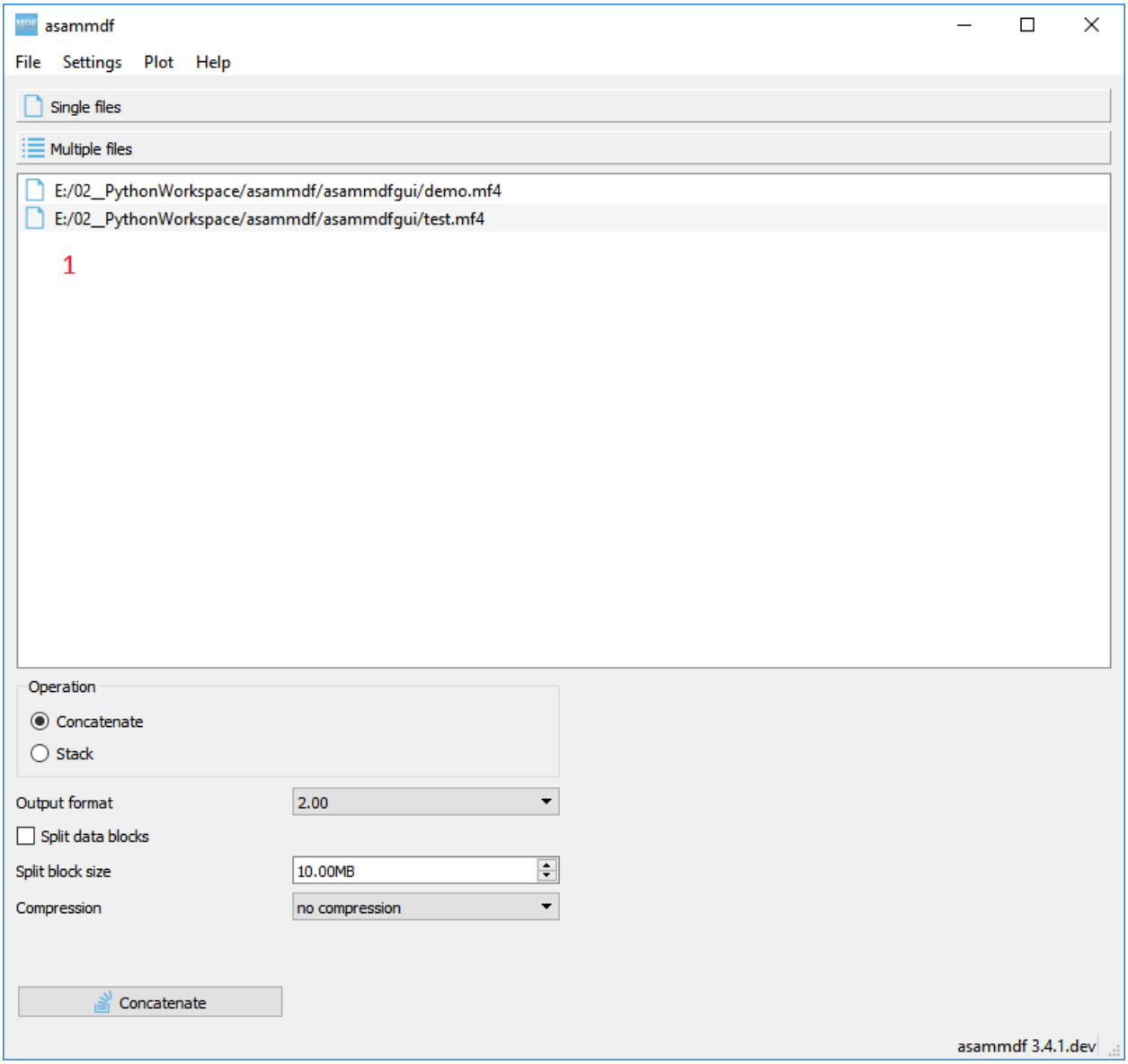


The *Statistics* panel is toggle using the *M* keyboard shortcut



7.3 Multiple files

The *Multiple files* toolbox page is used to concatenate or stack multiple files.



The files list can be rearanged in the list (1) by drag and dropping lines. Unwanted files can be deleted by selecting them and pressing the *DEL* key. The files order is considered from top to bottom.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

A

astype() (asammdf.signal.Signal method), 47
AttachmentBlock (class in asammdf.v4_blocks), 37

C

Channel (class in asammdf.v2_v3_blocks), 20
Channel (class in asammdf.v4_blocks), 38
ChannelConversion (class in asammdf.v2_v3_blocks), 21
ChannelConversion (class in asammdf.v4_blocks), 40
ChannelDependency (class in asammdf.v2_v3_blocks), 23
ChannelExtension (class in asammdf.v2_v3_blocks), 23
ChannelGroup (class in asammdf.v2_v3_blocks), 24
ChannelGroup (class in asammdf.v4_blocks), 41
concatenate() (asammdf.mdf.MDF static method), 8
convert() (asammdf.mdf.MDF method), 8
cut() (asammdf.mdf.MDF method), 8
cut() (asammdf.signal.Signal method), 47

D

DataBlock (class in asammdf.v4_blocks), 43
DataGroup (class in asammdf.v2_v3_blocks), 25
DataGroup (class in asammdf.v4_blocks), 42
DataList (class in asammdf.v4_blocks), 43

E

export() (asammdf.mdf.MDF method), 8
extend() (asammdf.signal.Signal method), 48

F

FileHistory (class in asammdf.v4_blocks), 46
FileIdentificationBlock (class in asammdf.v2_v3_blocks), 26
FileIdentificationBlock (class in asammdf.v4_blocks), 44
filter() (asammdf.mdf.MDF method), 9

H

HeaderBlock (class in asammdf.v2_v3_blocks), 26
HeaderBlock (class in asammdf.v4_blocks), 44

I

interp() (asammdf.signal.Signal method), 48
iter_channels() (asammdf.mdf.MDF method), 10
iter_get() (asammdf.mdf.MDF method), 10
iter_groups() (asammdf.mdf.MDF method), 11

M

MDF (class in asammdf.mdf), 7
merge() (asammdf.mdf.MDF static method), 11

P

physical() (asammdf.signal.Signal method), 48
plot() (asammdf.signal.Signal method), 48
ProgramBlock (class in asammdf.v2_v3_blocks), 27

R

resample() (asammdf.mdf.MDF method), 11

S

SampleReduction (class in asammdf.v2_v3_blocks), 28
scramble() (asammdf.mdf.MDF static method), 11
select() (asammdf.mdf.MDF method), 11
Signal (class in asammdf.signal), 47
SourceInformation (class in asammdf.v4_blocks), 45
stack() (asammdf.mdf.MDF static method), 12
start_time (asammdf.v4_blocks.HeaderBlock attribute), 45

T

TextBlock (class in asammdf.v2_v3_blocks), 28
TextBlock (class in asammdf.v4_blocks), 46
TriggerBlock (class in asammdf.v2_v3_blocks), 29

W

whereis() (asammdf.mdf.MDF method), 13