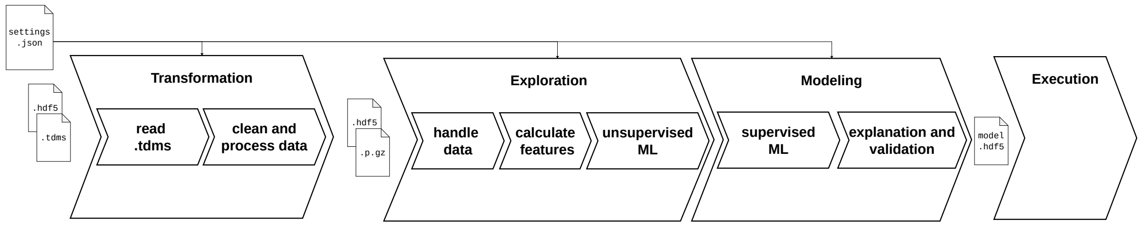
CERN

Transformation ML Framework

on basis of the XBox2 Data Set

Lorenz Fischl

Introduction



Introduction

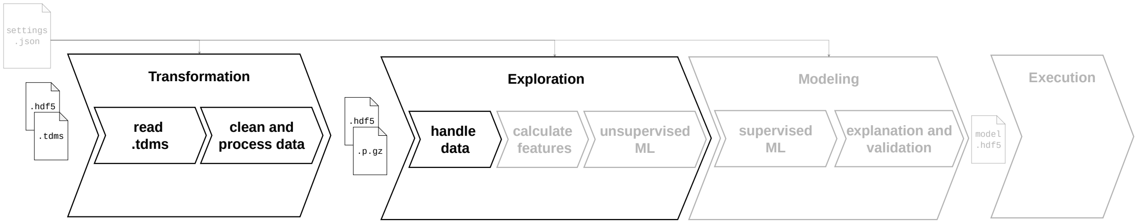


Table of Contents

Table of Contents

Table of Contents

.tdms files

.tdms files

- format by NI for well documented sensor data

`.tdms` files

- format by NI for well documented sensor data
- tree based consisting of the layers

.tdms files

- format by NI for well documented sensor data
- tree based consisting of the layers
  - the file

.tdms files

- format by NI for well documented sensor data
- tree based consisting of the layers
  - the file
  - groups

`.tdms` files

- format by NI for well documented sensor data
- tree based consisting of the layers
  - the file
  - groups
  - channels

.tdms files

- format by NI for well documented sensor data
- tree based consisting of the layers
    - the file
    - groups
    - channels
- properties make this format "well documented"

.tdms files

- format by NI for well documented sensor data
- tree based consisting of the layers
  - the file
  - groups
  - channels
- properties make this format ”well documented”

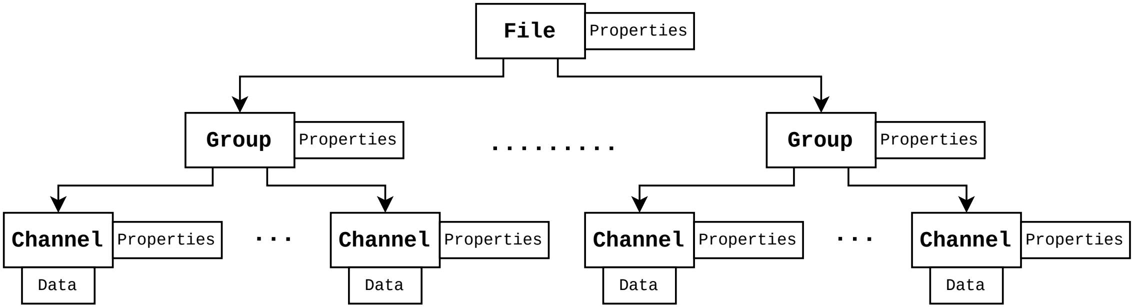


Table of Contents

XBox2 Data

XBox2 Data

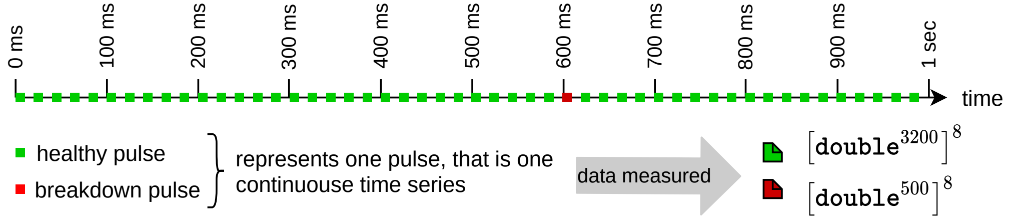
Every 20ms a pulse is sent into the RF cavity for particle acceleration.

XBox2 Data

Every 20*ms* a pulse is sent into the RF cavity for particle acceleration. Sometimes an arc forms. Those events are called breakdown.

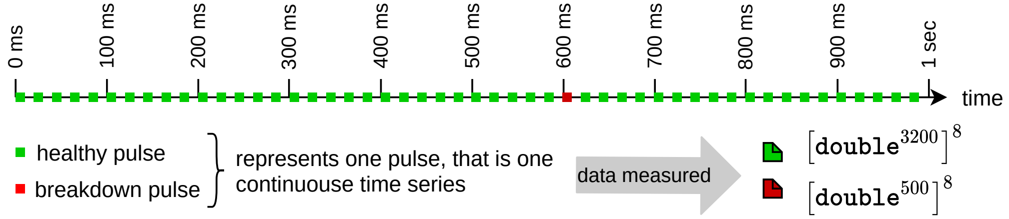
XBox2 Data

Every 20ms a pulse is sent into the RF cavity for particle acceleration. Sometimes an arc forms. Those events are called breakdown.


0 ms
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
800 ms
900 ms
1 sec
time
🟩 healthy pulse
🟥 breakdown pulse
represents one pulse, that is one continuouse time series
data measured
$\left[\mathtt{double}^{3200}\right]^8$
$\left[\mathtt{double}^{500}\right]^8$

XBox2 Data

Every 20*ms* a pulse is sent into the RF cavity for particle acceleration. Sometimes an arc forms. Those events are called breakdown.

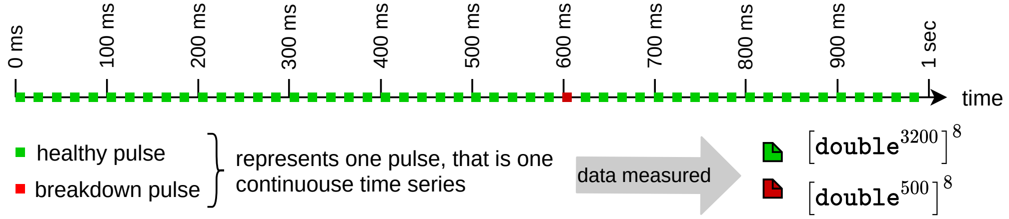


0 ms
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
800 ms
900 ms
1 sec
time
🟩 healthy pulse
🟥 breakdown pulse
represents one pulse, that is one continuouse time series
data measured
$\left[\mathtt{double}^{3200}\right]^8$
$\left[\mathtt{double}^{500}\right]^8$

temporary log files with 16 channels of 3200 and 500 sample points.

XBox2 Data

Every 20*ms* a pulse is sent into the RF cavity for particle acceleration. Sometimes an arc forms. Those events are called breakdown.


0 ms
100 ms
200 ms
300 ms
400 ms
500 ms
600 ms
700 ms
800 ms
900 ms
1 sec
time
healthy pulse
breakdown pulse
represents one pulse, that is one continuouse time series
data measured
$\left[\mathtt{double}^{3200}\right]^8$
$\left[\mathtt{double}^{500}\right]^8$

temporary log files with 16 channels of 3200 and 500 sample points.
storing all EventData would take up $\sim$ 1TB.

XBox2 EventData

XBox2 EventData

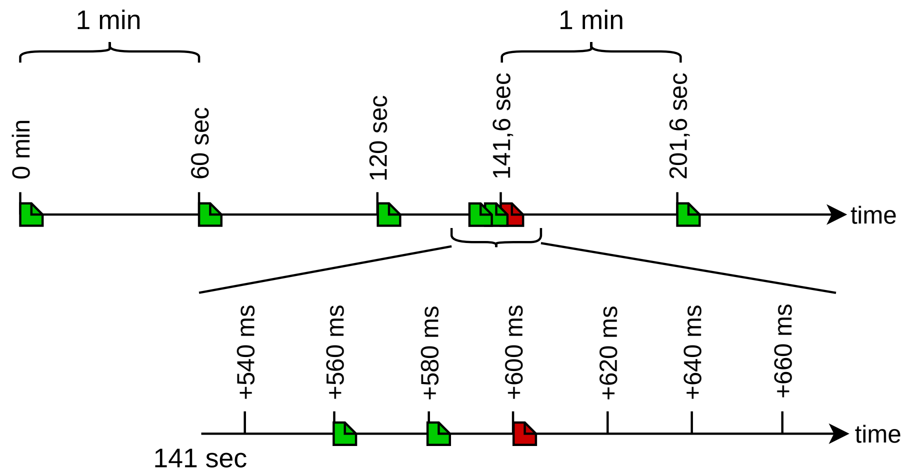
A log group of one pulse is stored every minute.

XBox2 EventData

A log group of one pulse is stored every minute.
When a breakdown happens the corresponding log group + the two prior log groups are stored.

XBox2 EventData

A log group of one pulse is stored every minute.
When a breakdown happens the corresponding log group + the two prior log groups are stored.



XBox2 TrendData

XBox2 TrendData

35 values about the environmental conditions (that don't change rapidly) are stored roughly every 1,5 sec.

XBox2 TrendData

35 values about the environmental conditions (that don't change rapidly) are stored roughly every 1,5 sec.
All TrendData of one day is stored in 1-2 groups.

XBox2 TrendData

35 values about the environmental conditions (that don't change rapidly) are stored roughly every 1,5 sec.
All TrendData of one day is stored in 1-2 groups.

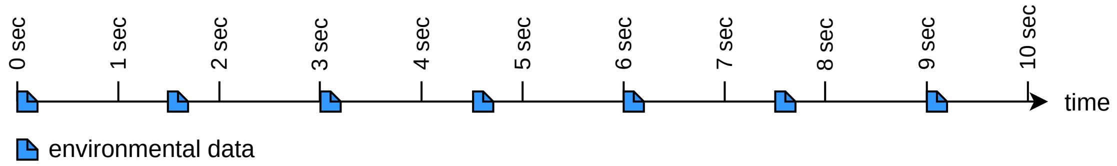


Table of Contents

Table of Contents

Requirements

We need a data format that

Requirements

We need a data format that

- is compatible between systems (SWAN, htcondor, locally)

Requirements

We need a data format that

- is compatible between systems (SWAN, htcondor, locally)
- can read data column wise fast

Requirements

We need a data format that

- is compatible between systems (SWAN, htcondor, locally)
- can read data column wise fast
- can be Transformed and handled (sorted/filtered/processed) relatively easy

Requirements

We need a data format that

- is compatible between systems (SWAN, htcondor, locally)
- can read data column wise fast
- can be Transformed and handled (sorted/filtered/processed) relatively easy
- can be transformed into `pandas` DataFrame

Requirements

We need a data format that

- is compatible between systems (SWAN, htcondor, locally)
- can read data column wise fast
- can be Transformed and handled (sorted/filtered/processed) relatively easy
- can be transformed into `pandas` DataFrame
- outputs `numpy` arrays for machine learning

Table of Contents

nptdms

`nptdms`

- package `nptdms` can read and handle `.tdms` files

nptdms

- package `nptdms` can read and handle `.tdms` files
- very slow (ex.: read of a 100MB file can take $>$30sec)

`nptdms`

- package `nptdms` can read and handle `.tdms` files
- very slow (ex.: read of a 100MB file can take $>$30sec)
- very space inefficient (ex. TrendData: 20,5 GB in .tdms $\rightarrow$ 2.8 GB of data)

Table of Contents

Transformation

Transformation

`.tdms`

Transformation

`.tdms`

`.tdms` file gets read with the `nptdms` package

Transformation

$\texttt{.tdms} \longrightarrow \texttt{pd df/ dictionary}$

data is stored in a `pandas` DataFrames (=pd df) and a python dictionary

Transformation

$$\texttt{.tdms} \longrightarrow \texttt{pd df/ dictionary}$$

data is stored in a `pandas` DataFrames (=pd df) and a python dictionary

- *EventData*: For each channel form all groups in one file, there is a single column pd df of vectors (channel wise reading).

Transformation

`.tdms` $\longrightarrow$ `pd df/ dictionary`

data is stored in a `pandas` DataFrames (=pd df) and a python dictionary

- *EventData*: For each channel form all groups in one file, there is a single column pd df of vectors (channel wise reading).
- *ContextData*: for each event (= group in the EventData) the prior TrendData and some context data (eg. pulse length/amplitude) is stored in a dictionary together with the EventData names.

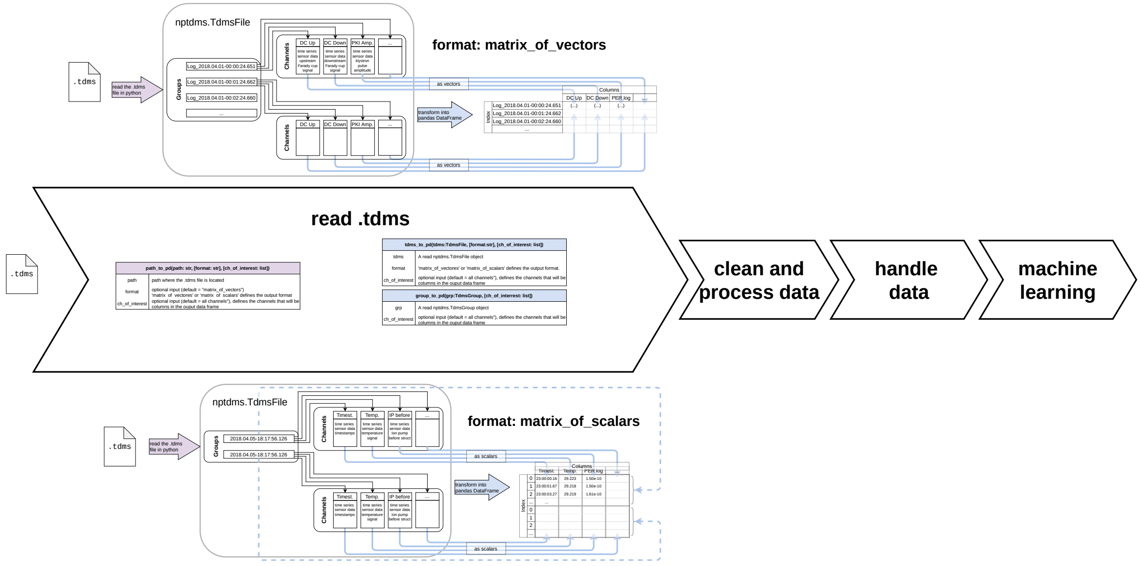
Transformation

$$\texttt{.tdms} \longrightarrow \texttt{pd df/ dictionary} \longrightarrow \texttt{pickle} \longrightarrow \texttt{.gzip}$$

data is serialized and compressed with the `compress_pickle` package

read .tdms

read .tdms



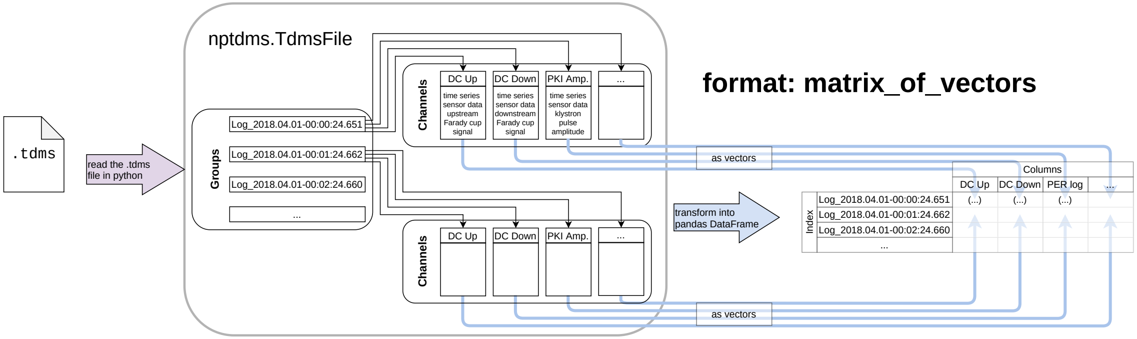
.tdms
read the .tdms file in python
nptdms.TdmsFile
Groups
Channels
Log_2018.04.01-00:00:24.651
Log_2018.04.01-00:01:24.662
Log_2018.04.01-00:02:24.660
...
DC Up
DC Down
PKI Amp.
time series sensor data upstream Faraday cup signal
time series sensor data downstream Faraday cup signal
time series sensor data klystron pulse amplitude
format: matrix_of_vectors
as vectors
transform into pandas DataFrame
Columns
DC Up
DC Down
PER log
Index
(...)
read .tdms
clean and process data
handle data
machine learning
nptdms.TdmsFile
2018.04.05-18:17:56.126
Timest.
Temp.
IP before
time series sensor data timestamps
time series sensor data temperature signal
time series sensor data Ion pump before struct
format: matrix_of_scalars
as scalars
Timest.
Temp.
PER log
23:00:00.16
29.223
1.50e-10
23:00:01.67
29.218
1.50e-10
23:00:03.27
29.219
1.61e-10

path_to_pd(path: str, [format: str], [ch_of_interest: list])	
path	path where the .tdms file is located
format	optional input (default = "matrix_of_vectors") 'matrix of vectores' or 'matrix of scalars' defines the output format
ch_of_interest	optional input (default = all channels"), defines the channels that will be columns in the ouput data frame

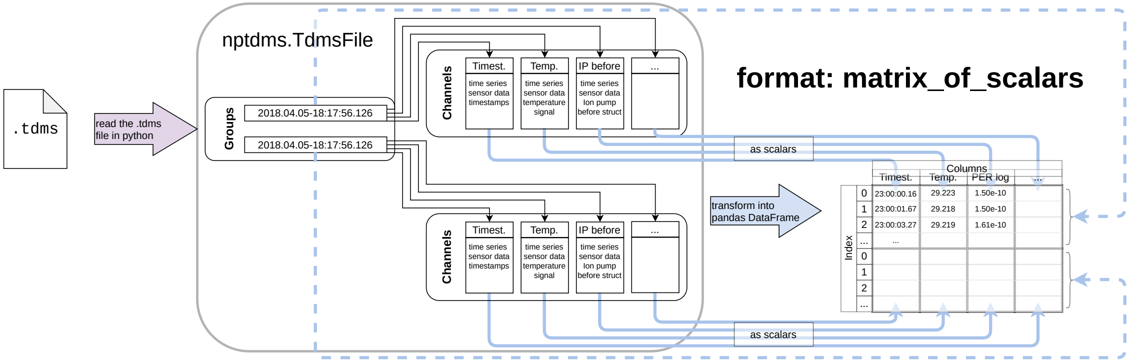
tdms_to_pd(tdms:TdmsFile, [format:str], [ch_of_interest: list])	
tdms	A read nptdms.TdmsFile object
format	'matrix_of_vectores' or 'matrix_of_scalars' defines the output format.
ch_of_interest	optional input (default = all channels"), defines the channels that will be columns in the ouput data frame

group_to_pd(grp:TdmsGroup, [ch_of_interrest: list])	
grp	A read nptdms.TdmsGroup object
ch_of_interest	optional input (default = all channels"), defines the channels that will be columns in the ouput data frame

format: matrix_of_vectors


.tdms
read the .tdms file in python
nptdms.TdmsFile
Groups
Log_2018.04.01-00:00:24.651
Log_2018.04.01-00:01:24.662
Log_2018.04.01-00:02:24.660
...
Channels
DC Up
time series sensor data upstream Farady cup signal
DC Down
time series sensor data downstream Farady cup signal
PKI Amp.
time series sensor data klystron pulse amplitude
...
as vectors
transform into pandas DataFrame
Columns
DC Up
DC Down
PER log
Index
(...)

format: matrix_of_scalars



Handling

The Data is handled (=sorted, filtered, etc.) in `pandas` DataFrames

Handling

The Data is handled (=sorted, filtered, etc.) in `pandas` DataFrames

- *EventData*: unzipped and unpickled with the `compress_pickle` package. (channel wise reading)

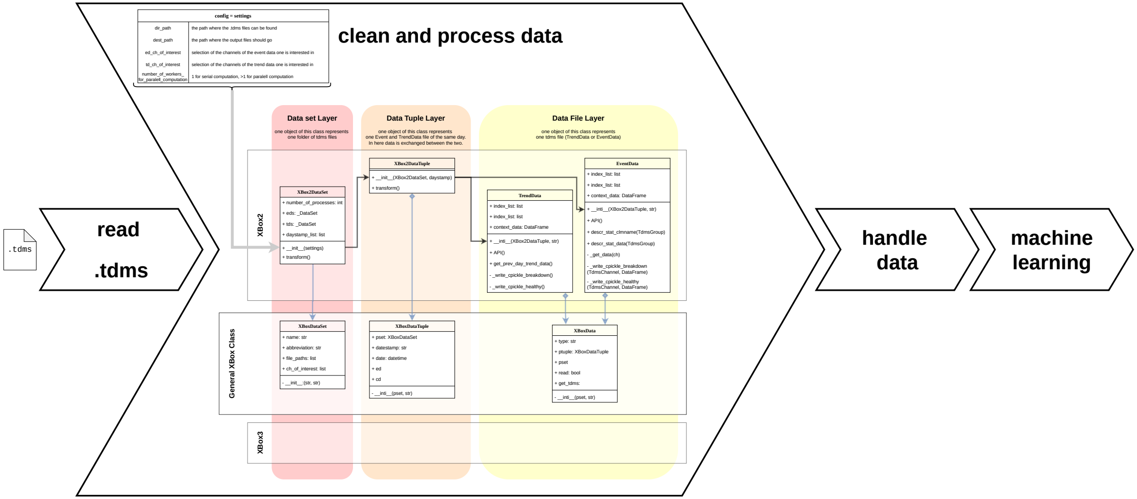
Handling

The Data is handled (=sorted, filtered, etc.) in `pandas` DataFrames

- *EventData*: unzipped and unpickled with the `compress_pickle` package. (channel wise reading)
- *ContextData*: the dictionary is also unzipped and unpickled and then transformed into a `pandas` DataFrame.

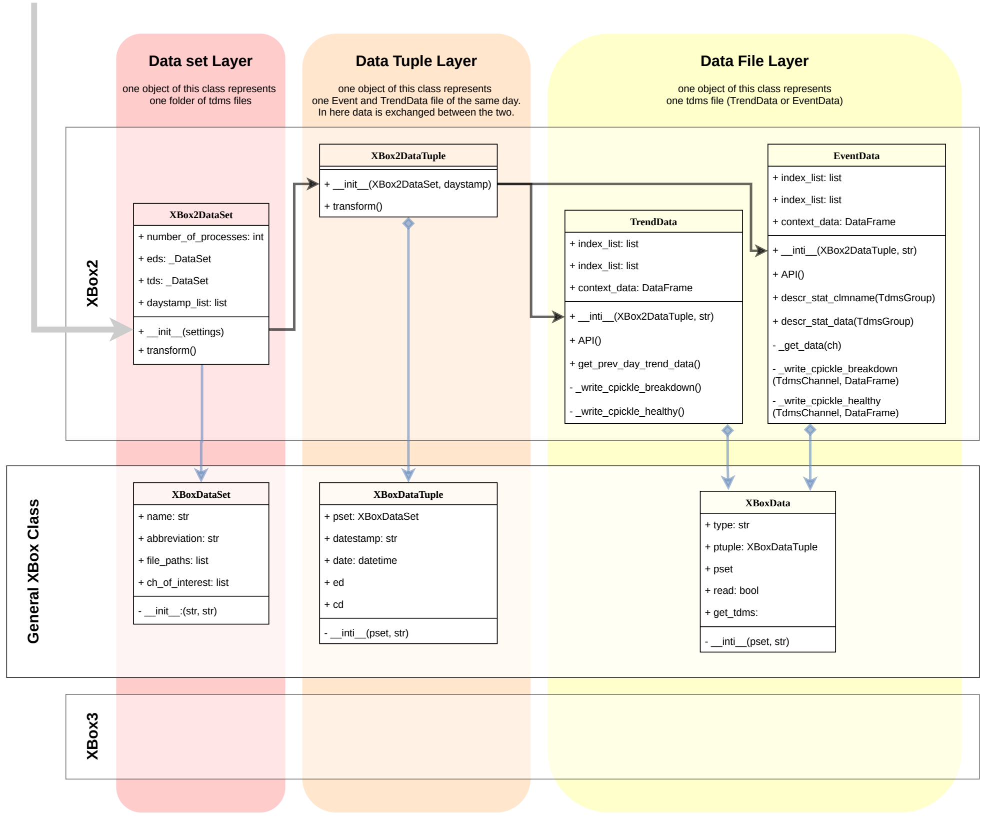
Clean and Process data with classes

Clean and Process data with classes



config = settings	
dir_path	the path where the .tdms files can be found
dest_path	the path where the output files should go
ed_ch_of_interest	selection of the channels of the event data one is interested in
td_ch_of_interest	selection of the channels of the trend data one is interested in
number_of_workers_ for_paralell_computation	1 for serial computation, >1 for paralell computation

.tdms
read
.tdms
clean and process data
Data set Layer
one object of this class represents one folder of tdms files
Data Tuple Layer
one object of this class represents one Event and TrendData file of the same day. In here data is exchanged between the two.
Data File Layer
one object of this class represents one tdms file (TrendData or EventData)
XBox2
XBox2DataSet
+ number_of_processes: int
+ eds: _DataSet
+ tds: _DataSet
+ daystamp_list: list
+ __init__(settings)
+ transform()
XBox2DataTuple
+ __init__(XBox2DataSet, daystamp)
+ transform()
TrendData
+ index_list: list
+ index_list: list
+ context_data: DataFrame
+ __inti__(XBox2DataTuple, str)
+ API()
+ get_prev_day_trend_data()
- _write_cpickle_breakdown()
- _write_cpickle_healthy()
EventData
+ index_list: list
+ index_list: list
+ context_data: DataFrame
+ __inti__(XBox2DataTuple, str)
+ API()
+ descr_stat_clmname(TdmsGroup)
+ descr_stat_data(TdmsGroup)
- _get_data(ch)
- _write_cpickle_breakdown (TdmsChannel, DataFrame)
- _write_cpickle_healthy (TdmsChannel, DataFrame)
General XBox Class
XBoxDataSet
+ name: str
+ abbreviation: str
+ file_paths: list
+ ch_of_interest: list
- __init__:(str, str)
XBoxDataTuple
+ pset: XBoxDataSet
+ datestamp: str
+ date: datetime
+ ed
+ cd
- __inti__(pset, str)
XBoxData
+ type: str
+ ptuple: XBoxDataTuple
+ pset
+ read: bool
+ get_tdms:
- __inti__(pset, str)
XBox3
handle data
machine learning


Data set Layer
one object of this class represents one folder of tdms files
Data Tuple Layer
one object of this class represents one Event and TrendData file of the same day. In here data is exchanged between the two.
Data File Layer
one object of this class represents one tdms file (TrendData or EventData)
XBox2
XBox2DataSet
+ number_of_processes: int
+ eds: _DataSet
+ tds: _DataSet
+ daystamp_list: list
+ __init__(settings)
+ transform()
XBox2DataTuple
+ __init__(XBox2DataSet, daystamp)
+ transform()
TrendData
+ index_list: list
+ index_list: list
+ context_data: DataFrame
+ __inti__(XBox2DataTuple, str)
+ API()
+ get_prev_day_trend_data()
- _write_cpickle_breakdown()
- _write_cpickle_healthy()
EventData
+ index_list: list
+ index_list: list
+ context_data: DataFrame
+ __inti__(XBox2DataTuple, str)
+ API()
+ descr_stat_clmname(TdmsGroup)
+ descr_stat_data(TdmsGroup)
- _get_data(ch)
- _write_cpickle_breakdown (TdmsChannel, DataFrame)
- _write_cpickle_healthy (TdmsChannel, DataFrame)
General XBox Class
XBoxDataSet
+ name: str
+ abbreviation: str
+ file_paths: list
+ ch_of_interest: list
- __init__:(str, str)
XBoxDataTuple
+ pset: XBoxDataSet
+ datestamp: str
+ date: datetime
+ ed
+ cd
- __inti__(pset, str)
XBoxData
+ type: str
+ ptuple: XBoxDataTuple
+ pset
+ read: bool
+ get_tdms:
- __inti__(pset, str)
XBox3

Summary

Summary

- pandas DataFrame are easy to use in notebooks

Summary

- pandas DataFrame are easy to use in notebooks
- pickle speeds up reading time

Summary

- `pandas` DataFrame are easy to use in notebooks
- pickle speeds up reading time
- compression takes up less space

Summary

- pandas DataFrame are easy to use in notebooks
- pickle speeds up reading time
- compression takes up less space

- version issues with pickle protocol

Summary

- pandas DataFrame are easy to use in notebooks
- pickle speeds up reading time
- compression takes up less space

- version issues with pickle protocol
- part of the process should not be in the Datahanlder instead of the Transformation

Summary

- `pandas` DataFrame are easy to use in notebooks
- pickle speeds up reading time
- compression takes up less space

- version issues with pickle protocol
- part of the process should not be in the Datahanlder instead of the Transformation
- EventData and TrenadData are stored differently

Summary

- `pandas` DataFrame are easy to use in notebooks
- pickle speeds up reading time
- compression takes up less space

- version issues with pickle protocol
- part of the process should not be in the Datahanlder instead of the Transformation
- EventData and TrenadData are stored differently
- channel properties are lost

Summary

- `pandas` DataFrame are easy to use in notebooks
- pickle speeds up reading time
- compression takes up less space

- version issues with pickle protocol
- part of the process should not be in the Datahanlder instead of the Transformation
- EventData and TrenadData are stored differently
- channel properties are lost
- data was changed in place in notebooks in retrospect

Table of Contents

.hdf5

.hdf5

- Hierarchical Data Format with a Unix file system like tree structure

`.hdf5`

- Hierarchical Data Format with a Unix file system like tree structure
- build for data management and storage

.hdf5

- Hierarchical Data Format with a Unix file system like tree structure
- build for data management and storage
- keys work like pointers to the memory, channel wise reading possible

`.hdf5`

- Hierarchical Data Format with a Unix file system like tree structure
- build for data management and storage
- keys work like pointers to the memory, channel wise reading possible
- in `nptdms` the function `to_hdf()` exists

`.hdf5`

- Hierarchical Data Format with a Unix file system like tree structure
- build for data management and storage
- keys work like pointers to the memory, channel wise reading possible
- in `nptdms` the function `to_hdf()` exists
- no data is lost in the Transformation, also properties are stored

Current Results with `.hdf5`

Current Results with `.hdf5`

- converted TrendData (20.5GB $\rightarrow$ 2.8GB) into the `.hdf5` files locally with 8 cores (using the `nptdms` function `tmds.to_hdf()`)

Current Results with `.hdf5`

- converted TrendData (20.5GB $\rightarrow$ 2.8GB) into the `.hdf5` files locally with 8 cores (using the `nptdms` function `tmds.to_hdf()`)
  - converting `tdms` to `.hdf5` took in total $\sim$ 1h

Current Results with `.hdf5`

- converted TrendData (20.5GB $\rightarrow$ 2.8GB) into the `.hdf5` files locally with 8 cores (using the `nptdms` function `tmds.to_hdf()`)
  - converting `tdms` to `.hdf5` took in total $\sim$ 1h
  - reading a sinlge channel from all data takes $<$ 0.5sec

Current Results with `.hdf5`

- converted TrendData (20.5GB $\rightarrow$ 2.8GB) into the `.hdf5` files locally with 8 cores (using the `nptdms` function `tmds.to_hdf()`)
  - converting `tdms` to `.hdf5` took in total $\sim$ 1h
  - reading a sinlge channel from all data takes $<$ 0.5sec
- converted EventData (73.3GB $\rightarrow$ 77.7GB) locally with 4 cores `tmds.to_hdf()`)

Current Results with `.hdf5`

- converted TrendData (20.5GB $\rightarrow$ 2.8GB) into the `.hdf5` files locally with 8 cores (using the `nptdms` function `tmds.to_hdf()`)
  - converting `tdms` to `.hdf5` took in total $\sim$ 1h
  - reading a sinlge channel from all data takes $<$ 0.5sec
- converted EventData (73.3GB $\rightarrow$ 77.7GB) locally with 4 cores `tmds.to_hdf()`)
  - converting `tdms` to `.hdf5` took in total $\sim$ 2h

Current Results with `.hdf5`

- converted TrendData (20.5GB $\rightarrow$ 2.8GB) into the `.hdf5` files locally with 8 cores (using the `nptdms` function `tmds.to_hdf()`)
  - converting `tdms` to `.hdf5` took in total $\sim$ 1h
  - reading a sinlge channel from all data takes $<$ 0.5sec
- converted EventData (73.3GB $\rightarrow$ 77.7GB) locally with 4 cores `tmds.to_hdf()`)
  - converting `tdms` to `.hdf5` took in total $\sim$ 2h
  - reading a sinlge channel from all data takes $<$ 0.5sec

Table of Contents

Conclusion

Conclusion

- I implemented a generic class for converting `.tdms` files into `pd.df+cpickle`

Conclusion

- I implemented a generic class for converting `.tdms` files into `pd.df+cpickle`
- is there a better data format, maybe `.hdf5`?

Conclusion

- I implemented a generic class for converting `.tdms` files into `pd.df+cpickle`
- is there a better data format, maybe `.hdf5`?

Conclusion

- I implemented a generic class for converting `.tdms` files into `pd.df+cpickle`
- is there a better data format, maybe `.hdf5`?

	`nptdms`	`pd.df+cpickle`		`.hdf5`	
		w/o zip	w zip	w/o zip	w/ zip
space (GByte)	20.5GB	2.8GB	1GB	2.8GB	1GB
read (TD 1 channel)	$\sim$ 60min	4sec	12sec	0.5 sec	
read (TD 3 channels)	$\sim$ 60min	4sec	12sec	1 sec	
read (TD 15 channels)	$\sim$ 60min	4sec	12sec	4 sec	
feature calc. (ED)	$>$ 15min		7sec	8 sec	

CERN

home.cern