# General Time Series Data Format

The General Time Series Data Format is a binary hdf5 data format for storing time series data.

Mads M Pedersen, [mmpe@dtu.dk](mailto:mmpe@dtu.dk), DTU-Wind Energy (AED)

Torben Juul Larsen

### Features:

* Single file
* Optional data type, e.g. 16bit integer (compact) or 64 bit floating point (high precision)
* Precise time representation (including absolute times)
* Additional data blocks can be appended continuously
* Optional specification of name and description of dataset
* Optional specification of name, unit and description of attributes
* NaN support

### File contents

|  |  |  |
| --- | --- | --- |
| Name in file | Hdf5 type | Description |
| type | Attribute | "General Time Series Data Format" |
| [name] | Attribute | Dataset name |
| [description] | Attribute | Dataset description |
| [attribute\_names] | Dataset (no\_attributes x 1) | Attribute names |
| [attribute\_units] | Dataset (no\_attributes x 1) | Attribute units |
| [attribute\_descriptions] | Dataset (no\_attributes x 1) | Attribute descriptions |
| no\_blocks | Attribute | Number of blocks in file |
| block0000 | Group | Data block group |
| [Block0001] | Group | Data block group |
| … |  |  |
| [blockxxxx] | Group | Data block group |

### Block contents

|  |  |  |
| --- | --- | --- |
| blockxxxx/data | Dataset  (no\_observations x no\_attributes) | Data values may be compressed using gain and offset |
| [blockxxxx/time] | Dataset  (no\_observations x 1) | Absolute or relative time (may be compressed by time\_step and time\_start) |
| [blockxxxx/time\_step] | Attribute | Real time (e.g. seconds) of one time unit |
| [blockxxxx/time\_start] | Attribute | Absolute or relative time start |
| [blockxxxx/gains] | Dataset  (no\_observations x 1) | Data scale factors |
| [blockxxxx/offsets] | Dataset  (no\_observations x 1) | Data offsets |

### How to save

|  |
| --- |
| **Required parameters** |
| Filename |
| Data (no\_obsercations x no\_attributes) |
| Data type (default uint16) |

|  |  |
| --- | --- |
| **Optional parameters** | **Check** |
| Dataset name |  |
| Dataset description |  |
| Attribute names | Len==no\_attributes |
| Attribute units | Len==no\_attributes |
| Attribute descriptions | Len==no\_attributes |
| Absolute or relative time | Len==no\_observations |
| Time step i.e. real time of one time unit |  |
| Time start |  |

**Procedure**

Create hdf5 file

save type, value="General Time Series Data Format"

create group "block0000"

if Data type is integer type then

offsets = ColumnSums(Data) #Ignore NaN

data = Data – offsets

gains = ColumnMax(data) / (MaxInt(Data type)-1) #Ignore NaN

data = data / gains # where gains > 0, Ignore NaN

convert data to dtype

set data to MaxInt(Data Type) where data==NaN

save data, gains and offsets in block0000-group

else

convert data to dtype

save data in block0000-group

end if

check and save present optional parameters

### How to append blocks

|  |
| --- |
| **Required parameters** |
| Filename |
| Data (no\_obsercations x no\_attributes) |

|  |  |
| --- | --- |
| **Optional parameters** | **Check** |
| Absolute or relative time | Len==no\_observations |
| Time step i.e. real time of one time unit |  |
| Time start |  |

**Procedure**

Open hdf5 file for append

Check lcase(file.type)="general time series data format"

blocknr = file.no\_blocks

file.no\_blocks = blocknr+1

create group "blockxxxx", e.g. "block0004"

dtype = file.block0000.data.dtype

if dtype is integer type then

offsets = ColumnSums(Data) #Ignore NaN

data = Data – offsets

gains = ColumnMax(data) / (MaxInt(Data type)-1) #Ignore NaN

data = data / gains # where gains > 0, Ignore NaN

convert data to dtype

set data to MaxInt(Data Type) where data==NaN

save data, gains and offsets in blockxxxx-group

else

convert data to dtype

save data in blockxxxx-group

end if

check and save present optional parameters in blockxxxx

### How to load

Default values

|  |  |
| --- | --- |
| Name in file | Default value if not present |
| type | Required!!! |
| data | Required!!! |
| name | <filename> |
| description | "" |
| attribute\_names | None |
| attribute\_units | None |
| attribute\_descriptions | None |
| time | 0..no\_observations-1 |
| time\_step | 1 |
| time\_start | 0 |
| gains | 1 |
| offsets | 0 |

Read values from file or defaults values

Check lcase(type) == "general time series data format"

data = []

time = []

for i = 0 to file.no\_blocks

block = file.blockxxxx

if block.dtype is integer then

set block = NaN where block==MaxInt(block.dtype)

end if

block\_data = block.data \* block.gains + block.offset

data.append(block\_data)

block.time = block.time \* block.time\_step + block.time\_start

time.append(block\_time)

return time, data, <optional values>

# Appendix 1 – Python implementation

*'''*

*Created on 12/09/2013*

*@author: Mads M. Pedersen (mmpe@dtu.dk)*

*'''*

from \_\_future\_\_ import division, print\_function, absolute\_import, unicode\_literals

import h5py

import os

try: range = xrange; xrange = None

except NameError: pass

try: str = unicode; unicode = None

except NameError: pass

import numpy as np

import numpy.ma as ma

block\_name\_fmt = *"block%04d"*

def **load**(filename, dtype=np.float32):

*"""*

*load a General Time Series Data Format - datafile*

*=================================================*

*Parameters*

*----------*

*filename : str or open h5py.File object*

*filename or open file object*

*dtype: numpy dtype*

*type of returned data array, e.g. float16, float32 or float64*

*Returns*

*-------*

*numpy array (dtype=float64, size=no\_observations)*

*time*

*numpy array (dtype=dtype, size = no\_observations x no\_attributes)*

*data*

*dict*

*info containing:*

*- type: "General Time Series Data Format"*

*- name: name of dataset or filename if not present in file*

*- [description]: description of dataset or "" if not present in file*

*- [attribute\_names]: list of attribute names*

*- [attribute\_units]: list of attribute units*

*- [attribute\_descriptions]: list of attribute descriptions*

*"""*

if isinstance(filename, h5py.File):

f = filename

filename = f.filename

else:

f = h5py.File(filename, *'r'*)

try:

info = dict(f.attrs.items())

check\_type(f)

if (block\_name\_fmt % 0) not in f:

raise ValueError(*"HDF5 file must contain a group named '%s'"* % (block\_name\_fmt % 0))

block0 = f[block\_name\_fmt % 0]

if *'data'* not in block0:

raise ValueError(*"group %s must contain a dataset called 'data'"* % (block\_name\_fmt % 0))

\_, no\_attributes = block0[*'data'*].shape

if *'name'* not in info:

info[*'name'*] = os.path.splitext(os.path.basename(filename))[0]

info[*'description'*] = f.attrs.get(*'description'*, *""*)

if *'attribute\_names'* in f:

info[*'attribute\_names'*] = f[*'attribute\_names'*][:]

if *'attribute\_units'* in f:

info[*'attribute\_units'*] = f[*'attribute\_units'*][:]

if *'attribute\_descriptions'* in f:

info[*'attribute\_descriptions'*] = f[*'attribute\_descriptions'*][:]

no\_blocks = f.attrs[*'no\_blocks'*]

data = np.empty((0, no\_attributes))

time = np.empty((0), dtype=np.float64)

for i in range(no\_blocks):

block = f[block\_name\_fmt % i]

no\_observations, no\_attributes = block[*'data'*].shape

block\_time = (block.get(*'time'*, np.arange(no\_observations))[:]).astype(np.float64)

if *'time\_step'* in block.attrs:

block\_time \*= block.attrs[*'time\_step'*]

if *'time\_start'* in block.attrs:

block\_time += block.attrs[*'time\_start'*]

time = np.append(time, block\_time)

block\_data = block[*'data'*][:].astype(dtype)

if *"int"* in str(block[*'data'*].dtype):

block\_data[block\_data == np.iinfo(block[*'data'*].dtype).max] = np.nan

if *'gains'* in block:

block\_data \*= block[*'gains'*][:]

if *'offsets'* in block:

block\_data += block[*'offsets'*][:]

data = np.append(data, block\_data, 0)

f.close()

return time, data.astype(dtype), info

except (ValueError, AssertionError):

f.close()

raise

def **save**(filename, data, \*\*kwargs):

*"""*

*Save a General Time Series Data Format - datafile*

*=================================================*

*Parameters*

*----------*

*- filename*

*- data [numpy array size no\_observations x no\_attributes]*

*- kwargs \*optional\* arguments:*

*- name [str]*

*- description [str]*

*- attribute\_names [list with no\_attributes strings]*

*- attribute\_units [list with no\_attributes strings]*

*- attribute\_descriptions [list with no\_attributes strings]*

*- time [numpy array size no\_observations], default=0..no\_observations-1*

*- time\_step (e.g. 1/sample frequency), [int or float type], default=1*

*- time\_start (e.g. start time in seconds since 1/1/1970), [int or float type], default=0*

*- dtype [numpy.dtype], data type of saved data array, default uint16*

*"""*

if not filename.lower().endswith(*'.hdf5'*):

filename += *".hdf5"*

f = h5py.File(filename, *"w"*)

try:

f.attrs[*"type"*] = *"General time series data format"*

no\_observations, no\_attributes = data.shape

if *'name'* in kwargs:

f.attrs[*'name'*] = kwargs[*'name'*]

if *'description'* in kwargs:

f.attrs[*'description'*] = kwargs[*'description'*]

f.attrs[*'no\_attributes'*] = no\_attributes

if *'attribute\_names'* in kwargs:

assert(len(kwargs[*'attribute\_names'*]) == no\_attributes)

f.create\_dataset(*"attribute\_names"*, data=np.array(kwargs[*'attribute\_names'*], dtype=np.string\_))

if *'attribute\_units'* in kwargs:

assert(len(kwargs[*'attribute\_units'*]) == no\_attributes)

f.create\_dataset(*"attribute\_units"*, data=np.array(kwargs[*'attribute\_units'*], dtype=np.string\_))

if *'attribute\_descriptions'* in kwargs:

assert(len(kwargs[*'attribute\_descriptions'*]) == no\_attributes)

f.create\_dataset(*"attribute\_descriptions"*, data=np.array(kwargs[*'attribute\_descriptions'*], dtype=np.string\_))

f.attrs[*'no\_blocks'*] = 0

f.close()

append\_block(filename, data, \*\*kwargs)

except AssertionError:

f.close()

raise

def **append\_block**(filename, data, \*\*kwargs):

try:

f = h5py.File(filename, *"a"*)

check\_type(f)

no\_observations, no\_attributes = data.shape

assert(no\_attributes == f.attrs[*'no\_attributes'*])

blocknr = f.attrs[*'no\_blocks'*]

if blocknr == 0:

dtype = kwargs.get(*'dtype'*, np.uint16)

else:

dtype = f[block\_name\_fmt % 0][*'data'*].dtype

block = f.create\_group(block\_name\_fmt % blocknr)

if *'time'* in kwargs:

assert(len(kwargs[*'time'*]) == no\_observations)

block.create\_dataset(*'time'*, data=kwargs[*'time'*])

if *'time\_step'* in kwargs:

time\_step = kwargs[*'time\_step'*]

block.attrs[*'time\_step'*] = time\_step

if *'time\_start'* in kwargs:

block.attrs[*'time\_start'*] = kwargs[*'time\_start'*]

if *"int"* in str(dtype):

nan = np.isnan(data)

non\_nan\_data = ma.masked\_array(data, nan)

offsets = np.min(non\_nan\_data, 0)

data = np.copy(data)

data -= offsets

gains = np.max(non\_nan\_data - offsets, 0).astype(np.float64) / (np.iinfo(dtype).max - 1) #-1 to save value for NaN

not0 = np.where(gains != 0)

data[:, not0] /= gains[not0]

data = data.astype(dtype)

data[nan] = np.iinfo(dtype).max

block.create\_dataset(*'gains'*, data=gains)

block.create\_dataset(*'offsets'*, data=offsets)

block.create\_dataset(*"data"*, data=data.astype(dtype))

f.attrs[*'no\_blocks'*] = blocknr + 1

f.close()

except AssertionError:

f.close()

raise

def **check\_type**(f):

if *'type'* not in f.attrs or f.attrs[*'type'*].lower() != *"general time series data format"*:

raise ValueError(*"HDF5 file must contain a 'type'-attribute with the value 'General time series data format'"*)

if *'no\_blocks'* not in f.attrs:

raise ValueError(*"HDF5 file must contain an attribute named 'no\_blocks'"*)

# Appendix 2 – MatLab implementation

function [time, data, info] = gtsdf\_load(filename)

if nargin==0

filename = *'examples/all.hdf5'*;

end

%h5disp(*'examples/minimum.hdf5'*);

%info = h5info(filename);

function value = att\_value(name, addr, default)

try

value = h5readatt(filename, addr,name);

catch

if nargin==3

value = default;

else

value = *''*;

end

end

end

function r = read\_dataset(name, addr, default)

try

r = h5read(filename, strcat(addr,name));

catch

r = default;

end

end

if not (strcmpi(att\_value(*'type'*,*'/'*), *'general time series data format'*))

error(*'HDF5 file must contain a ''type''-attribute with the value ''General time series data format'''*)

end

if strcmp(att\_value(*'no\_blocks'*,*'/'*),*''*)

error(*'HDF5 file must contain an attribute named ''no\_blocks'''*)

end

hdf5info = h5info(filename);

if not (strcmp(hdf5info.Groups(1).Name,*'/block0000'*))

error(*'HDF5 file must contain a group named ''block0000'''*)

end

datainfo = h5info(filename,*'/block0000/data'*);

no\_attributes = datainfo.Dataspace.Size(1);

type = att\_value(*'type'*,*'/'*);

name = att\_value(*'name'*, *'/'*,*'no\_name'*);

description = att\_value(*'description'*, *'/'*);

attribute\_names = read\_dataset(*'attribute\_names'*,*'/'*, {});

attribute\_units = read\_dataset(*'attribute\_units'*,*'/'*, {});

attribute\_descriptions = read\_dataset(*'attribute\_descriptions'*,*'/'*, {});

info = struct(*'type'*,type, *'name'*, name, *'description'*, description, *'attribute\_names'*, {attribute\_names}, *'attribute\_units'*, {attribute\_units}, *'attribute\_descriptions'*,{attribute\_descriptions});

no\_blocks = att\_value(*'no\_blocks'*,*'/'*);

time = [];

data = [];

for i=0:no\_blocks-1

blockname = num2str(i,*'/block%04d/'*);

blokdatainfo = h5info(filename,strcat(blockname,*'data'*));

no\_observations = datainfo.Dataspace.Size(2);

blocktime = double(read\_dataset(*'time'*, blockname, [0:no\_observations-1]*'));*

blocktime\_start = att\_value(*'time\_start'*,blockname,0);

blocktime\_step = att\_value(*'time\_step'*,blockname,1);

time = [time;(blocktime\*blocktime\_step) + double(blocktime\_start)];

block\_data = read\_dataset(*'data'*, blockname)*';*

if isinteger(block\_data)

nan\_pos = block\_data==intmax(class(block\_data));

block\_data = double(block\_data);

block\_data(nan\_pos) = nan;

gains = double(read\_dataset(*'gains'*,blockname,1.));

offsets = double(read\_dataset(*'offsets'*, blockname,0));

for c = 1:no\_attributes

block\_data(:,c) = block\_data(:,c)\*gains(c)+offsets(c);

end

end

data = [data;block\_data];

end

end