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, DTU-Wind Energy (AED)
Torben Juul Larsen, DTU-Wind Energy (AED)

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
- Supports NAN (not a number)

File contents

Fields in square brackets, e.g. [name], are optional

Name in file	Hdf5 type	Description
type	Attribute	Must be "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

Group contents

Name in file	Hdf5 type	Description
group.data	Dataset (no_observations x no_attributes)	Data values may be compressed using gain and offset
[group.time]	Dataset (no_observations x 1)	Absolute or relative time (may be compressed by time_step and time_start)
[group.time_step]	Attribute	Real time (e.g. seconds) of one time unit
[group.time_start]	Attribute	Absolute or relative time start
[group.gains]	Dataset (no_observations x 1)	Data scale factors
[group.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	
Time start	

Procedure

```
Create hdf5 file
file.type = "General Time Series Data Format"
file.no_blocks = 1
create group "block0000"
if Data type is integer type then
            offsets = ColumnMin(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 Data type
            where data==NaN set data to MaxInt(Data Type)
            group.data = data
            group.offsets = offsets
            group.gains = gains
else
            convert data to Data type
            group.data = data
end if
check present optional fields

if present set:
            group.time = absolute or relative time
            group.time_step = Time step
            group.time_start = Time start
            file.name = Dataset name
            file.description = Dataset description
            file.attribute_names = Attribute names
            file.attribute_units = Attribute units
            file.attribute_descriptions = Attribute descriptions
```

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	
Time start	

Procedure

```
Open hdf5 file for append
Check lcase(file.type)="general time series data format"
blocknr = file.no_blocks
create group "block%4d"%blocknr, e.g. "block0001"
file.no_blocks = blocknr+1
dtype = file.block0000.data.dtype
if dtype is integer type then
            offsets = ColumnMin(Data) #Ignore NaN
            data = Data – offsets
            gains = ColumnMax(data) / (MaxInt(dtype)-1) #Ignore NaN
            data = data / gains # where gains > 0, Ignore NaN
            convert data to dtype
            where data==NaN set data to MaxInt(dtype)
            group.data = data
            save data, gains and offsets in block0000-group
else
            convert data to dtype
            group.data = data
end if

check present optional fields

if present set:
            group.time = absolute or relative time
            group.time_step = Time step
            group.time_start = Time start
```

How to load

Default values

Field in file	**Default value if not present**
file.type	Required!!!
file.data	Required!!!
file.name	<filename>
file.description	""
file.attribute_names	None
file.attribute_units	None
file.attribute_descriptions	None
group.time	0..no_observations-1
group.time_step	1
group.time_start	0
group.gains	1
group.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
            group = file.”block%4d”%i, e.g. “block0000”
            if group.dtype is integer then
                        set block = NaN where group == MaxInt(group.dtype)
            end if
            block_data = group.data * group.gains + group.offset
            data.append(block_data)
            block_time = group.time * group.time_step + group.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
```