phconvert Documentation

Release 0.7.2+0.g42d7ece

Antonino Ingargiola

Apr 13, 2017

Contents

1 Module hdf5 3

2 Module loader 7

3 Module pqreader 9

4 Module bhreader 11

5 Indices and tables 15

Python Module Index 17

Version 0.7.2+0.g42d7ece (release notes)

phconvert is a python 2 & 3 library which helps writing valid Photon-HDF5 files. This document contains the API documentation for phconvert.

The phconvert library contains two main modules: *hdf5* and *loader*. The former contains functions to save and validate Photon-HDF5 files. The latter, contains functions to load other formats to be converted to Photon-HDF5.

The phconvert repository contains a set the notebooks to convert existing formats to Photon-HDF5 or to write Photon-HDF5 from scratch:

- phconvert notebooks (read online).

In particular see notebook Writing Photon-HDF5 files (read online) as an example of writing Photon-HDF5 files from scratch.

Finally, phconvert repository contains a JSON specification of the Photon-HDF5 format which lists all the valid field names and corresponding data types and descriptions.

Contents:

CHAPTER 1

Module hdf5

The module *hdf5* defines functions to save and validate Photon-HDF5 files. The main two functions in this module are:

- *save_photon_hdf5()* to saves data from a dictionary to Photon-HDF5.
- *assert_valid_photon_hdf5()* to validate if a HDF5 file is valid Photon-HDF5.

This module also provides functions to save free-form dict to HDF5 (*dict_to_group()*) and read a HDF5 group into a dict (*dict_from_group()*). Finally there are utility functions to easily print HDF5 nodes and attributes (*print_children()*, *print_attrs()*).

For more info see: Writing Photon-HDF5 files.

List of functions

Main functions to save and validate Photon-HDF5 files.

phconvert.hdf5.**save_photon_hdf5**(*data_dict*, *h5_fname=None*, *user_descr=None*, *overwrite=False*, *compression={'complib': 'zlib', 'complevel': 6}*, *close=True*, *validate=True*, *warnings=True*, *skip_measurement_specs=False*, *require_setup=True*, *debug=False*)

Saves the dict *data_dict* in the Photon-HDF5 format.

This function requires the data to be saved as `data_dict` argument. The data needs to have the hierarchical structure of a Photon-HDF5 file. For the purpose, we use a standard python dictionary: each keys is a Photon-HDF5 field name and each value contains data (e.g. array, string, etc..) or another dictionary (in which case, it represents an HDF5 sub-group). Similarly, sub-dictionaries contain data or other dictionaries, as needed to represent the hierarchy of Photon-HDF5 files.

Features of this function:

•Checks that all field names are valid Photon-HDF5 field names.

•Checks that all field type match the Photon-HDF5 specs (scalar, array, or string).

- Populates automatically the identity group with filename, software, version and file creation date.
- Populates automatically the provenance group with info on the original data file (if it can be found on disk): creation and modification date, path.
- Computes field *acquisition_duration* when not provided (single-spot data only).

Minimal fields required to create a Photon-HDF5 file:

- */description* (string)
- */photon_data/timestamps* (array)
- */photon_data/timestamps_specs/timestamps_unit* (scalar float)
- */setup/num_pixels* (int): number of detectors
- */setup/num_spots* (int): number of excitation/detection spots
- */setup/num_spectral_ch* (int): number of detection spectral bands
- */setup/num_polarization_ch* (int): number of detected polarization states
- */setup/num_split_ch* (int): number of beam splitted channels
- */setup/modulated_excitation* (bool): True if excitation is alternated.
- */setup/lifetime* (bool): True if dataset contains TCSPC data.

See also Writing Photon-HDF5 files.

As a side effect *data_dict* is modified by adding the key '_data_file' containing a reference to the pytables file.

Parameters

- **data_dict** (*dict*) – the dictionary containing the photon data. The keys must strings matching valid Photon-HDF5 paths. The values must be scalars, arrays, strings or another dict.
- **h5_fname** (*string or None*) – file name for the output Photon-HDF5 file. If None, the file name is taken from `data_dict['_filename']` with extension changed to '.hdf5'.
- **user_descr** (*dict or None*) – dictionary of descriptions (strings) for user-defined fields. The keys must be strings representing the full HDF5 path of each field. The values must be binary (i.e. encoded) strings restricted to the ASCII set.
- **overwrite** (*bool*) – if True, a pre-existing HDF5 file with same name is overwritten. If False, save the new file by adding the suffix "new_copy" (and if a "_new_copy" file is already present overwrites it).
- **compression** (*dict*) – a dictionary containing the compression type and level. Passed to pytables *tables.Filters()*.
- **close** (*bool*) – If True (default) the HDF5 file is closed before returning. If False the file is left open.
- **validate** (*bool*) – if True, after saving perform a validation step raising an error if the specs are not followed.
- **warnings** (*bool*) – if True, print warnings for important optional fields that are missing. If False, don't print warnings.
- **skip_measurement_specs** (*bool*) – if True don't print any warning for missing measurement_specs group.

- **require_setup** (*bool*) – if True, raises an error if some mandatory fields in /setup are missing. If False, allows missing setup fields (or missing setup altogether). Use False when saving only detectors' dark counts.
- **debug** (*bool*) – if True prints additional debug information.

For description and specs of the Photon-HDF5 format see: http://photon-hdf5.readthedocs.org/

phconvert.hdf5.**assert_valid_photon_hdf5**(*datafile*, *warnings=True*, *verbose=False*, *strict_description=True*, *require_setup=True*, *skip_measurement_specs=False*)

Asserts that `datafile` follows the Photon-HDF5 specs.

If the input datafile does not follow the specifications, it raises the `Invalid_PhotonHDF5` exception, with a message indicating the cause of the error.

This function checks that:

•all fields are valid Photon-HDF5 names

•all fields have valid descriptions

•all mandatory fields are present

•if /setup/lifetime is True (i.e. 1), assures that nanotimes and nanotimes_specs are present

Parameters

- **datafile** (*string or tables.File*) – input data file to be validated
- **warnings** (*bool*) – if True, print warnings for important optional fields that are missing. If False, don't print warnings.
- **verbose** (*bool*) – if True print details about the performed tests.
- **strict_description** (*bool*) – if True consider a non-conforming description (TITLE) a specs violation.
- **require_setup** (*bool*) – if True, raises an error if some mandatory fields in /setup are missing. If False, allows missing setup fields (or missing setup altogether).
- **skip_measurement_specs** (*bool*) – if True don't print any warning for missing measurement_specs group.

Utility functions

Utility functions to work with HDF5 files in pytables.

phconvert.hdf5.**print_children**(*group*)

Print all the sub-groups in *group* and leaf-nodes children of *group*.

Parameters **group** (*pytables group*) – the group to be printed.

phconvert.hdf5.**print_attrs**(*node*, *which='user'*)

Print the HDF5 attributes for *node_name*.

Parameters

- **node** (*pytables node*) – node whose attributes will be printed. Can be either a group or a leaf-node.
- **which** (*string*) – Valid values are 'user' for user-defined attributes, 'sys' for pytables-specific attributes and 'all' to print both groups of attributes. Default 'user'.

phconvert.hdf5.**dict_from_group**(*group*, *read=True*)
Return a dict with the content of a PyTables *group*.

phconvert.hdf5.**dict_to_group**(*group*, *dictionary*)
Save *dictionary* into HDF5 format in *group*.

CHAPTER 2

Module `loader`

This module contains functions to load each supported data format. Each loader function loads data from a third-party formats into a python dictionary which has the structure of a Photon-HDF5 file. These dictionaries can be passed to *phconvert.hdf5.save_photon_hdf5()* to save the data in Photon-HDF5 format.

The loader module contains high-level functions which "fill" the dictionary with the appropriate arrays. The actual decoding of the input binary files is performed by low-level functions in other modules (smreader.py, pqreader.py, bhreader.py). When trying to decode a new file format, these modules can provide useful examples.

phconvert.loader.**nsalex_bh**(*filename_spc*, *donor=4*, *acceptor=6*, *alex_period_donor=(10*, *1500)*, *alex_period_acceptor=(2000*, *3500)*, *excitation_wavelengths=(5.32e-07*, *6.35e-07)*, *detection_wavelengths=(5.8e-07*, *6.8e-07)*, *allow_missing_set=False*, *tcspc_num_bins=None*, *tcspc_unit=None*)

Load a .spc and (optionally) .set files for ns-ALEX and return 2 dict.

The first dictionary can be passed to the *phconvert.hdf5.save_photon_hdf5()* function to save the data in Photon-HDF5 format.

Returns the first contains the main photon data (timestamps, detectors, nanotime, ...); the second contains the raw data from the .set file (it can be saved in a user group in Photon-HDF5).

Return type Two dictionaries

phconvert.loader.**nsalex_ht3**(*filename*, *donor=0*, *acceptor=1*, *alex_period_donor=(150*, *1500)*, *alex_period_acceptor=(1540*, *3050)*, *excitation_wavelengths=(5.23e-07*, *6.28e-07)*, *detection_wavelengths=(5.8e-07*, *6.8e-07)*)

Load a .ht3 file containing ns-ALEX data and return a dict.

This dictionary can be passed to the *phconvert.hdf5.save_photon_hdf5()* function to save the data in Photon-HDF5 format.

phconvert.loader.**usalex_sm**(*filename*, *donor=0*, *acceptor=1*, *alex_period=4000*, *alex_offset=750*, *alex_period_donor=(2850*, *580)*, *alex_period_acceptor=(930*, *2580)*, *excitation_wavelengths=(5.32e-07*, *6.35e-07)*, *detection_wavelengths=(5.8e-07*, *6.8e-07)*, *software='LabVIEW Data Acquisition usALEX'*)

Load a .sm us-ALEX file and returns a dictionary.

This dictionary can be passed to the *phconvert.hdf5.save_photon_hdf5()* function to save the data in Photon-HDF5 format.

CHAPTER 3

Module `pqreader`

This module contains functions to load and decode files from PicoQuant hardware.

The primary exported functions are:

- *`load_ht3()`* which returns decoded timestamps, detectors, nanotimes and metadata from an HT3 file.
- *`load_pt3()`* which returns decoded timestamps, detectors, nanotimes and metadata from a PT3 file.

Other lower level functions are:

- *`ht3_reader()`* which loads metadata and raw t3 records from HT3 files
- *`pt3_reader()`* which loads metadata and raw t3 records from PT3 files
- *`process_t3records()`* which decodes the t3 records returning timestamps (after overflow correction), detectors and TCSPC nanotimes.

Note that the functions performing overflow/rollover correction can take advantage of numba, if installed, to significanly speed-up the processing.

List of functions

High-level functions to load and decode several PicoQuant file formats:

`phconvert.pqreader.`**`load_ht3`**(*filename*, *ovcfunc=None*)

Load data from a PicoQuant .ht3 file.

Parameters

- **`filename`** (*`string`*) – the path of the HT3 file to be loaded.
- **`ovcfunc`** (*`function or None`*) – function to use for overflow/rollover correction of timestamps. If None, it defaults to the fastest available implementation for the current machine.

Returns A tuple of timestamps, detectors, nanotimes (integer arrays) and a dictionary with metadata containing at least the keys 'timestamps_unit' and 'nanotimes_unit'.

phconvert.pqreader.**load_pt3**(*filename*, *ovcfunc=None*)
Load data from a PicoQuant .pt3 file.

Parameters

- **filename** (*string*) – the path of the PT3 file to be loaded.
- **ovcfunc** (*function or None*) – function to use for overflow/rollover correction of timestamps. If None, it defaults to the fastest available implementation for the current machine.

Returns A tuple of timestamps, detectors, nanotimes (integer arrays) and a dictionary with metadata containing at least the keys 'timestamps_unit' and 'nanotimes_unit'.

Low-level functions

These functions are the building blocks for loading and decoding the different files:

phconvert.pqreader.**ht3_reader**(*filename*)
Load raw t3 records and metadata from an HT3 file.

phconvert.pqreader.**pt3_reader**(*filename*)
Load raw t3 records and metadata from a PT3 file.

phconvert.pqreader.**process_t3records**(*t3records*, *time_bit=10*, *dtime_bit=15*, *ch_bit=6*, *special_bit=True*, *ovcfunc=None*)
Extract the different fields from the raw t3records array (.ht3).

Returns 3 arrays representing detectors, timestamps and nanotimes.

CHAPTER 4

Module `bhreader`

This module contains functions to load and decode files from Becker & Hickl hardware.

The high-level function in this module are:

- *`load_spc()`* which loads and decoded the photon data from SPC files.
- *`load_set()`* which returns a dictionary of metadata from SET files.

Becker & Hickl SPC Format

The structure of the SPC format is here described.

SPC-600/630

SPC-600/630 files have a record of 48-bit (6 bytes) in little endian (<) format. The first 6 bytes of the file are an header containing the *timestamps_unit* (in 0.1ns units) in the two central bytes (i.e. bytes 2 and 3). In the following drawing each char represents 2 bits:

```
bit: 64        48                          0
     0000 0000 XXXX XXXX XXXX XXXX XXXX XXXX
               '_______' '__' '__'   '_____'
field names:       a      c    b        d

     0000 0000 XXXX XXXX XXXX XXXX XXXX XXXX
               '_______' '__' '__' '_______'
numpy dtype:       a      c    b    field0

macrotime = [ b  ] [     a     ]  (24 bit)
detector  = [ c  ]                (8 bit)
nanotime  = [  d  ]               (12 bit)

overflow bit: 13, bit_mask = 2^(13-1) = 4096
```

SPC-134/144/154/830

SPC-134/144/154/830 files have a record of 32-bits (4 bytes) in little endian (<) format. The first 4 bytes of the file are an header containing the *timestamps_unit* (in 0.1ns units) in first two bytes. In the following drawing each char represents 2 bits:

```
bit:                      32                0
                          XXXX XXXX XXXX XXXX
                          '''_____' '''_____'
field names:              a    b    c    d

                          XXXX XXXX XXXX XXXX
                          '_______' '_______'
numpy dtype:               field1    field0

macrotime = [ d ]        (12 bit)
detector  = [ c ]        (4 bit)
nanotime  = [ b ]        (12 bit)
aux       = [ a ]        (4 bit)

aux = [invalid, overflow, gap, mark]

If overflow == 1 and invalid == 1 --> number of overflows = [ b ][ c ][ d ]
```

List of functions

High-level functions to load and decode Becker & Hickl SPC/SET pair of files:

phconvert.bhreader.**load_spc**(*fname*, *spc_model='SPC-630'*)
Load data from Becker & Hickl SPC files.

Parameters **spc_model** (*string*) – name of the board model. Valid values are 'SPC-630', 'SPC-134', 'SPC-144', 'SPC-154' and 'SPC-830'.

Returns 3 numpy arrays (timestamps, detector, nanotime) and a float (timestamps_unit).

phconvert.bhreader.**load_set**(*fname_set*)
Return a dict with data from the Becker & Hickl .SET file.

Low-level functions

These functions are the building blocks for decoding Becker & Hickl files:

phconvert.bhreader.**bh_set_identification**(*fname_set*)
Return a dict containing the IDENTIFICATION section of .SET files.

The both keys and values are native strings (binary strings on py2 and unicode strings on py3).

phconvert.bhreader.**bh_set_sys_params**(*fname_set*)
Return a dict containing the SYS_PARAMS section of .SET files.

The keys are native strings (traditional strings on py2 and unicode strings on py3) while values are numerical type or byte strings.

phconvert.bhreader.**bh_decode**(*s*)
Replace code strings from .SET files with human readble label strings.

phconvert.bhreader.**bh_print_sys_params**(*sys_params*)
Print a summary of the Becker & Hickl system parameters (.SET file).

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

phconvert.bhreader, 11
phconvert.hdf5, 3
phconvert.loader, 7
phconvert.pqreader, 9

Index

A

assert_valid_photon_hdf5() (in module phconvert.hdf5), 5

B

bh_decode() (in module phconvert.bhreader), 12
bh_print_sys_params() (in module phconvert.bhreader), 12
bh_set_identification() (in module phconvert.bhreader), 12
bh_set_sys_params() (in module phconvert.bhreader), 12

D

dict_from_group() (in module phconvert.hdf5), 5
dict_to_group() (in module phconvert.hdf5), 6

H

ht3_reader() (in module phconvert.pqreader), 10

L

load_ht3() (in module phconvert.pqreader), 9
load_pt3() (in module phconvert.pqreader), 9
load_set() (in module phconvert.bhreader), 12
load_spc() (in module phconvert.bhreader), 12

N

nsalex_bh() (in module phconvert.loader), 7
nsalex_ht3() (in module phconvert.loader), 7

P

phconvert.bhreader (module), 11
phconvert.hdf5 (module), 3
phconvert.loader (module), 7
phconvert.pqreader (module), 9
print_attrs() (in module phconvert.hdf5), 5
print_children() (in module phconvert.hdf5), 5
process_t3records() (in module phconvert.pqreader), 10
pt3_reader() (in module phconvert.pqreader), 10

S

save_photon_hdf5() (in module phconvert.hdf5), 3

U

usalex_sm() (in module phconvert.loader), 7