Photon-HDF5 Documentation

Release 0.4

Antonino Ingargiola, Xavier Michalet, Ted Laurence

Aug 30, 2017

Contents

Authors Antonino Ingargiola, Xavier Michalet, Ted Laurence

Contact Photon-HDF5 Google Group

Format Version 0.4

This is the reference documentation for **Photon-HDF5** (homepage), a file format for timestamp-based single-molecule spectroscopy experiments such as single-molecule FRET (smFRET) (with or without lifetime), Fluorescence Correlation Spectroscopy (FCS) and other related techniques.

Any dataset containing photon timestamps and other per-photon data can be stored in Photon-HDF5 files. Photon-HDF5 is designed for long-term preservation and data sharing and can store experimental details and metadata such as setup configurations, sample information, authorship and provenance.

The present document contains the reference documentation for the Photon-HDF5 format.

Other related resources (see also Photon-HDF5 Homepage):

- , view them with .
-
- : reference library to create and convert Photon-HDF5 files
- Biophysical Journal (2016), ().
-

CHAPTER 1

Table of Contents

Introduction

Overview

This document contains specifications of the Photon-HDF5 format. This format allows saving single-molecule spectroscopy experiment data in which at least one stream of photon timestamps is present. It has been designed as a standard container format for a broad range of experiments involving confocal microscopy. Examples are confocal smFRET experiments performed with a single or multiple excitation spots. Both μs-ALEX and ns-ALEX data are supported.

What problems are we trying to solve?

- Ensuring long term data persistence.
- A disk space and read speed efficient file format for repeated use as well as for archiving.
- Facilitating data sharing and interoperability between analysis programs and research groups.

Features of HDF5

- Open-standard: language and platform independent, self-describing format with open source implementations.
- Efficient: the HDF5 format is a binary format that allows compression and fast read/write operations.
- Flexible: data arrays can be stored in "groups" (hierarchical format). Metadata can be attached to each data entry (attributes).
- No limit in data size.
- Support for a variety of numeric and non-numeric data types.

Photon-HDF5: Design principles

The main design principles are:

- Simplicity
- Flexibility
- Compatibility

We aim to define a format that has a minimal set of specifications and therefore is easy to implement. At the same time, it is important that the format can be expanded to accommodate new use cases while maintaining backward compatibility.

To achieve simplicity, the only required file characteristics are a general file layout and the presence of a few basic attributes and parameters. The remaining (small set of) fields defined in this document will be present only when needed by a particular measurement.

We retain flexibility by allowing the user to save any arbitrary data outside the specs of this document. To assure that future versions of this format will not conflict with user-defined fields, we require that all user-defined fields be contained in groups called `user`.

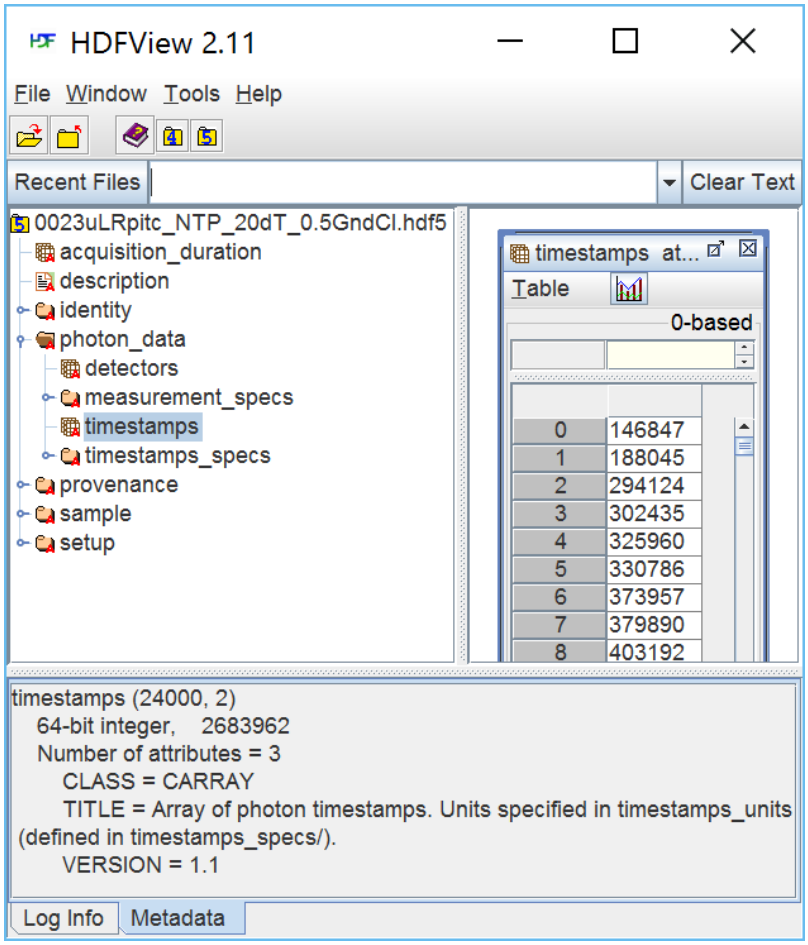
Photon-HDF5 format definition

Contents

Overview

A Photon-HDF5 is a HDF5 file with a predefined structure for timestamp-based data.

A screen-shot of a typical Photon-HDF5 file opened in HDFView is shown here:

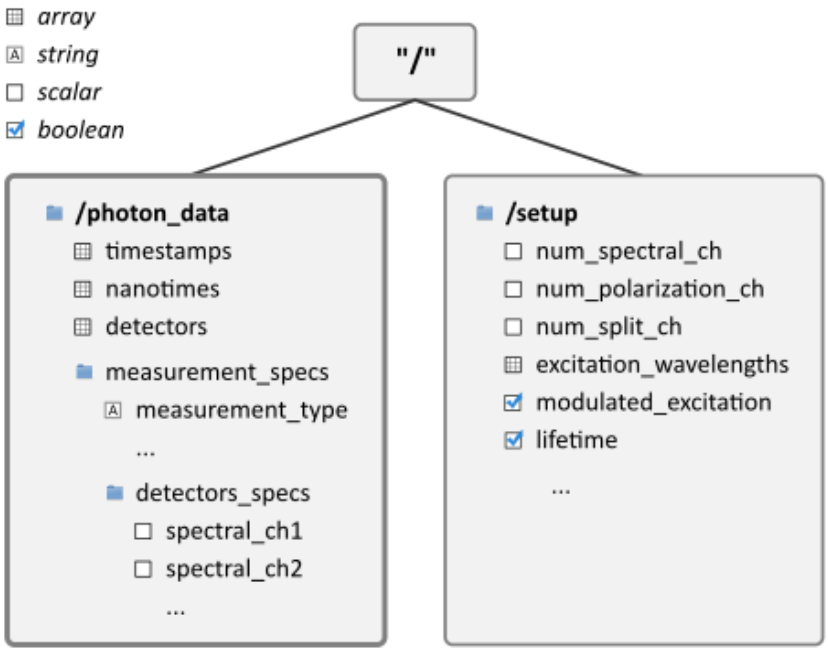


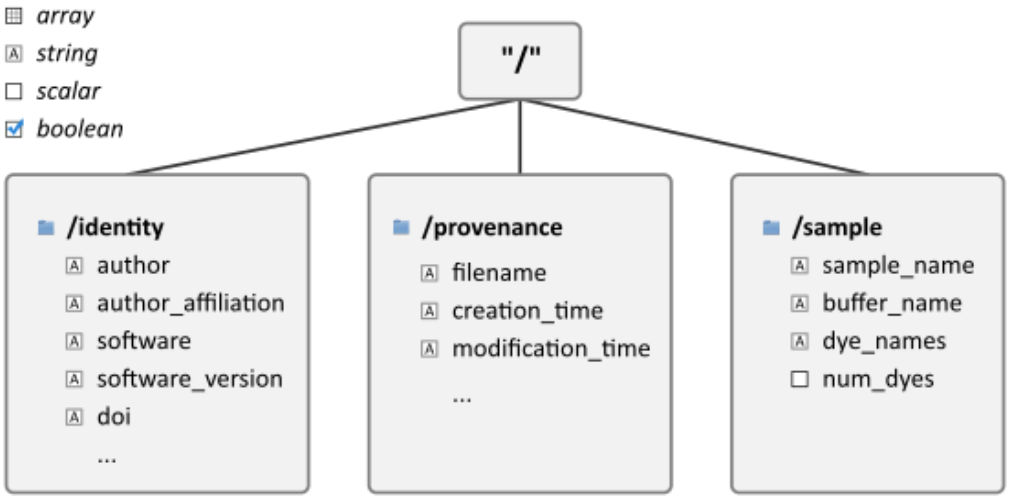
The previous figure shows the 5 main groups contained in a Photon-HDF5 file. Of these, */photon_data* and */setup* contains the raw data and all the information needed for the analysis. A schematic overview is shown in the next figure:

The remaining 3 groups provide additional metadata not necessary for data analysis:

A brief description of these 3 metadata groups follows:

- */identity*: Information about the data file.
- */provenance*: Information about the original data file (when the Photon-HDF5 file has been converted from another format).
- */sample*: Description of the measured sample.


array
string
scalar
boolean
"/"
/photon_data
timestamps
nanotimes
detectors
measurement_specs
measurement_type
...
detectors_specs
spectral_ch1
spectral_ch2
...
/setup
num_spectral_ch
num_polarization_ch
num_split_ch
excitation_wavelengths
modulated_excitation
lifetime
...


array
string
scalar
boolean
"/"
/identity
author
author_affiliation
software
software_version
doi
...
/provenance
filename
creation_time
modification_time
...
/sample
sample_name
buffer_name
dye_names
num_dyes

Root-level parameters

The root node ("/") in a Photon-HDF5 file contains the following fields:

- **/acquisition_duration**: (float) the measurement duration in seconds.
- **/description**: (string) a user-defined measurement description.

In addition, the root node has the following attributes which distinguish a Photon-HDF5 file from other HDF5 files:

- **format_name**: must contain the string "Photon-HDF5"
- **format_version**: (string) the Photon-HDF5 format version.

Photon-data group

This section describes the layout and fields in the **/photon_data** group. Note that only the kind of data is specified (i.e. scalar, integer array, float array), but no data type size is mandated. For arrays, the most commonly used data-type is indicated.

Mandatory fields:

- **timestamps**: (array) photon timestamps. Typical data-type int64.
- **timestamps_specs/**
  - **timestamps_unit**: (float) timestamp units in *seconds*.

Optional if there is only 1 detector, otherwise mandatory:

- **detectors**: (array of integers) *Detector pixel IDs* for each timestamp. Typical data-type uint8.

When the dataset contains TCSPC or nanotime information (i.e. arrival time of each photon with respect to a laser pulse), the following fields must be present:

- **nanotimes**:(array of integers) TCSPC nanotimes. Conventionally the time axis direction is the "natural" direction, i.e. lifetime decays look correctly oriented in time. For more details see *Nanotimes time direction*. Typical data-type uint16.
- **nanotimes_specs/**
  - **tcspc_unit**: (float) TAC/TDC bin size (in seconds).
  - **tcspc_num_bins**: (integer) number of TAC/TDC bins.
  - **tcspc_range**:(float) (optional) full-scale range of the TAC/TDC (in seconds). This a derived field equal to `tcspc_unit * tcspc_num_bins`.

Finally, if the data come from a simulation, `/photon_data` may contain:

- **particles**: (array of integers) a particle ID (integer) for each timestamp. Typical data-type uint8.

Measurement specs

The optional **/photon_data/measurement_specs** group contains additional information allowing unambiguous interpretation of the data for each specific type of measurement.

- **measurement_type**: (string) the type of the measurements. Valid names are:
  - "smFRET" (1 excitation color, 2 detection colors)
  - "smFRET-usALEX" (2 excitation colors, 2 detection colors)
  - "smFRET-usALEX-3c" (3 excitation colors, 3 detection colors)

  - "smFRET-nsALEX" (2 excitation colors, 2 detection colors)

  New names can be created for different kind of measurements and we encourage users to submit new name requests.

The *measurement_type* field describes the type of measurement saved within the file. It is an important field allowing software packages reading and saving Photon-HDF5 files to perform consistency checks (see also *Measurement type*).

For μs-ALEX, 2, 3 or N colors:

- **alex_period**: (integer or float) duration of one complete excitation alternation period expressed in timestamp units. The alternation period is equal to `alex_period * timestamps_unit`.

For ns-ALEX (or lifetime with no alternation):

- **laser_repetition_rate**: (float) excitation laser pulse repetition rate in *Hertz*.

For 2-color (or more) μs-ALEX and ns-ALEX (optional):

- **alex_offset**: (scalar) [μs-ALEX only] Time offset (in timestamps units) to be applied to the timestamps array before computing the μs-ALEX histogram. It is assumed that the μs-ALEX alternation histogram is the histogram of (`timestamps - alex_offset`) **MOD** `alex_period`.
- **alex_excitation_period1**: (array with an even-number of integer elements, normally 2) start and stop values identifying the excitation periods for the **first** wavelength in `/setup/excitation_wavelengths` (which is the shortest wavelength). In smFRET experiments with 2-colors excitation this field defines the *donor excitation period*. See also *Wavelengths and spectral band order* and note below.
- **alex_excitation_period2**: (array with an even-number of integer elements, normally 2) start and stop values identifying the excitation periods for the **second** wavelength in `/setup/excitation_wavelengths`. In smFRET experiments with 2-colors excitation this field defines the *acceptor excitation period*. See also *Wavelengths and spectral band order* and note below.

For 3 (or more) colors alternated or interleaved excitation:

- **alex_excitation_period3**: (array with an even-number of integer elements, normally 2) start and stop values identifying the excitation periods for the **third** wavelength in `/setup/excitation_wavelengths`. See also *Wavelengths and spectral band order* and note below.
- etc...

Note: For μs-ALEX, both *alex_excitation_period1* and *alex_excitation_period2* are 2-element arrays and are expressed in *timestamps_units*. For ns-ALEX (also known as PIE), they are arrays with an even-number of elements, comprising as many start-stop nanotime pairs as there are excitation periods within the TAC/TDC range. In this case the values are expressed in *nanotimes_units*.

For more details see *Definition of alternation periods*.

Detectors specs

Within **measurement_specs**, the **detectors_specs/** sub-group contains all the *pixel ID*–detection channel associations, i.e. spectral bands, polarizations or *beam-split channels*.

When a measurement records more than 1 spectral band, the fields:

- **spectral_ch1**
- **spectral_ch2**
- etc...

specify which detector pixel is employed in each spectral band. When the measurement records only 1 spectral band these fields may be omitted. The spectral bands are strictly ordered for increasing wavelengths. For example, for 2-color smFRET measurements `spectral_ch1` and `spectral_ch2` represent the *donor* and *acceptor* channel respectively.

If a measurement records more than 1 polarization states, the fields:

- **polarization_ch1**
- **polarization_ch2**

specify which detector pixel is used for each polarization. When the measurement records only one polarization, these fields may be omitted.

When the detection light is split into 2 channels using a non-polarizing beam-splitter the fields:

- **split_ch1**
- **split_ch2**

specify which detector pixel is used in each of the "beam-split" channels.

All previous fields are arrays containing one or more *Detector pixel IDs*. For example, a 2-color smFRET measurement will have only one value in `spectral_ch1` (donor) and one value in `spectral_ch2` (acceptor). A 2-color smFRET measurement with polarization (4 detectors) will have 2 values in each of the `spectral_chX` and `polarization_chX` fields (where X=1 or 2). For a multispot smFRET measurement, in each `photon_dataN` group, there will be `spectral_chX` fields containing the donor/acceptor pixels used in that spot (see *Multi-spot measurements*).

Setup group

The **/setup** group contains information about the measurement setup. This group can be **absent** in some files, an example being a file containing only detector dark counts, for which the following fields do not necessarily have a meaning. When setup is present, the following 7 fields are mandatory:

- **num_pixels**: (integer) total number of detector pixels. For example, for a single-spot 2-color smFRET measurement using 2 single-pixel SPADs as detectors this field is 2.
- **num_spots**: (integer) the number of excitation (or detection) "spots" in the sample. This field is 1 for all the measurements using a single confocal excitation volume. When not applicable, for example under wide-field illumination with 2-D imaging detectors, this field is omitted.
- **num_spectral_ch**: (integer) number of distinct detection spectral channels. For example, in a 2-color smFRET experiment there are 2 detection spectral channels (donor and acceptor), therefore its value is 2. When there is a single detection channel or all channels detect the same spectral band, its value is 1.
- **num_polarization_ch**: (integer) number of distinct detection polarization channels. For example, in polarization anisotropy measurements, its value is 2. When there is a single detection channel or all channels detect the same polarization (including when no polarization selection is performed) its value is 1.
- **num_split_ch**: (integer) number of distinct detection channels detecting the same spectral band **and** polarization state. For example, when a non-polarizing beam-splitter is employed in the detection path, its value is 2. When no splitting is performed, its value is 1.
- **modulated_excitation**: (boolean) *True* (or 1) if there is any form of excitation modulation either in the wavelength space (as in μs-ALEX or PAX) or in the polarization space. This field is also *True* for pulse-interleaved excitation (PIE) or ns-ALEX measurements.
- **lifetime**: (boolean) *True* (or 1) if the measurements includes a *nanotimes* array of (usually sub-ns resolution) photon arrival times with respect to a laser pulse (as in TCSPC measurements).

The remaining fields are optional:

- **excitation_wavelengths**: (array of floats) list of excitation wavelengths (center wavelength if broad-band) in increasing order (unit: *meter*).
- **excitation_cw**: (array of booleans) for each excitation source, this field indicates whether excitation is continuous wave (CW), *True*, or pulsed, *False*. The order of excitation sources is the same as that in `excitation_wavelengths` and is in increasing order of wavelengths.

The following fields are optional and not necessarily relevant for all experiments. If the associated information is irrelevant or not available, these fields are omitted.

- **excitation_polarizations**: (arrays of floats) list of polarization angles (in degrees) for each excitation source. The order of excitation sources is the same as in `excitation_wavelengths` and is in increasing order of wavelengths.
- **excitation_input_powers**: (array of floats) excitation power in *Watts* for each excitation source. This is the excitation power entering the optical system.
- **excitation_intensity**: (array of floats) excitation intensity in the sample for each excitation source (units: *Watts/meters*^{2}). In the case of confocal excitation this is the peak PSF intensity.
- **detection_wavelengths**: (arrays of floats) reference wavelengths (in *meters*) for each detection spectral band. This array is ordered in increasing order of wavelengths. The first element refers to `detectors_specs/spectral_ch1`, the second to `detectors_specs/spectral_ch2` and so on.
- **detection_polarizations**: (arrays of floats) polarization angles for each detection polarization band. The first element refers to `detectors_specs/polarization_ch1`, the second to `detectors_specs/polarization_ch2` and so on. This field is not relevant if no polarization selection is performed.
- **detection_split_ch_ratios**: (array of floats) power fraction detected by each "beam-split" channel (i.e. independent detection channels obtained through a non-polarizing beam splitter). For 2 beam-split channels that receive the same power this array should be `[0.5, 0.5]`. The first element refers to `detectors_specs/split_ch1`, the second to `detectors_specs/split_ch2` and so on. This field is not relevant when no polarization- and spectral-insensitive splitting is performed.

Sample group

The **/sample** group contains information related to the measured sample. This group is optional.

- **num_dyes**: (integer) number of different dyes present in the samples.
- **dye_names**: (string) comma-separated list of dye or fluorophore names (for example: `"ATTO550, ATTO647N"`)
- **buffer_name**: (string) a user defined description for the buffer.
- **sample_name**: (string) a user defined description for the sample.

Identity group

The **identity/** group contains information about the specific Photon-HDF5 file.

The following fields are mandatory (and automatically added by phconvert):

- **creation_time**: (string) the Photon-HDF5 file creation time with the following format: "YYYY-MM-DD HH:MM:SS".
- **software**: (string) name of the software used to create the Photon-HDF5 file.
- **software_version**: (string) version of the software used to create the Photon-HDF5 file.
- **format_name**: (string) this must always be "Photon-HDF5"

- **format_version**: (string) the Photon-HDF5 version string (e.g. "0.4")
- **format_url**: (string) A URL pointing to the Photon-HDF5 specification document.

The following fields are optional:

- **author**: (string) the author of the measurement (or simulation).
- **author_affiliation**: (string) the company or institution the *author* is affiliated with.
- **creator**: (string) the Photon-HDF5 file creator. Used when the data was previously stored in another format and the conversion is performed by a different person than the author.
- **creator_affiliation**: (string) the company or institution the *creator* is affiliated with.
- **url**: (string) URL that allow to download the Photon-HDF5 data file.
- **doi**: (string) Digital Object Identifier (DOI) for the Photon-HDF5 data file.
- **funding**: (string) Description of funding sources and or grants that supported the data collection.
- **license**: (string) License under which the data is released. Many journals and funding agencies require or suggest "CC0" (or equivalently "Public Domain") for the data.
- **filename**: (string) Photon-HDF5 file name at creation time. This field saves the original file name even if the file is later on renamed on disk.
- **filename_full**: (string) Photon-HDF5 file name (including the full path) at creation time.

Provenance group

The **provenance/** group contains info about the original file that has been converted into a Photon-HDF5 file. If the file is directly saved to Photon-HDF5 there is no previous "original" file and in this case the provenance group may be omitted. Also, if some information is not available the relative field may be omitted.

- **filename**: (string) File name of the original data file before conversion to Photon-HDF5.
- **filename_full**: (string) File name (with full path) of the original data file before conversion to Photon-HDF5.
- **creation_time**: (string) Creation time of the original data file.
- **modification_time**: (string) Time of last modification of the original data file.
- **software**: (string) Software used to save the original data file.
- **software_version**: (string) Version of the software used to save the original data file.

Additional notes and definitions

Detector pixel IDs

A *detector pixel ID* (or simply *pixel ID*) is the "name" of each pixels and is usually a single integer. Pixels are normally numbered incrementally, but not necessarily so. In other words, a file containing data taken with 2 single-point (pixel) detectors could have the first detector labeled "4" and the second detector labeled "6". In some cases (when using detector arrays) the pixel ID can be a *n*-tuple of integers. This allow to specify, for each pixel, the module number and the X, Y location, for example. Therefore, an array of pixel IDs can be either a 1-D column array or a 2-D array. In either cases, each row identifies a pixel.

Beam-split channels

When the emitted light path is split in 2 or more detection paths by using a non-polarizing beam splitter the measurement has so called beam-split channels. The fields *split_ch1* and *split_ch2* contains the list of *Detector pixel IDs* for each beam-split channel (see *Detectors specs*).

Beam split channels can receive same or different (depending on whether the beam splitter is 50-50). The fractional power of each beam split channel can be saved in the field *detection_split_ch_ratios* in the *Setup group*.

Wavelengths and spectral band order

In Photon-HDF5, by convention, all the excitation wavelengths and detection spectral bands are ordered in increasing order: from the shortest to the longest wavelength. This ordering is strictly followed and removes any ambiguity in defining first, second, etc... wavelength or spectral band.

For examples, for μs-ALEX and ns-ALEX (or PIE) the excitation wavelengths (in `/setup/excitation_wavelenths`) are ordered as

1. *donor excitation wavelength*,
2. *acceptor excitation wavelength*

Similarly, the donor (or acceptor) excitation period range is defined by `/photon_data/measurement_specs/alex_excitation_period1` (or `/photon_data/measurement_specs/alex_excitation_period2`).

Finally the donor (or acceptor) *Detector pixel IDs* number is defined in `/photon_data/measurement_specs/detectors_specs/spectral_ch1` (or `/photon_data/measurement_specs/detectors_specs/spectral_ch2`).

Definition of alternation periods

Note for μs-ALEX

The fields *alex_offset*, *alex_excitation_period1* and *alex_excitation_period2* define the excitation period for each excitation source. The alternation histogram is the histogram of the following quantity:

`A` = (`timestamps` - `alex_offset`) **MODULO** `alex_period`

Note that `alex_offset` must be a value that shifts the timestamps in a way that the resulting alternation histogram has uninterrupted excitation periods for each excitation source. It can be thought as the delay between the start of the timestamping and the start of the alternation modulation. In most cases this is just an empirical parameter depending on the specific setup.

Photons emitted during the donor period (or, respectively, acceptor period) are obtained by applying the condition:

- `(A >= start) and (A < stop)`

Measurement type

Each *measurement_type* has an associated set of mandatory fields which must be present to ensure that all information needed to unambiguously interpret the data is present. For example, for a 2-color smFRET measurement, a software package creating a file should check that the association between detector-pixel and donor or acceptor channel is present. If some necessary field is absent, the software package should warn the user in order that this information is added before saving the file.

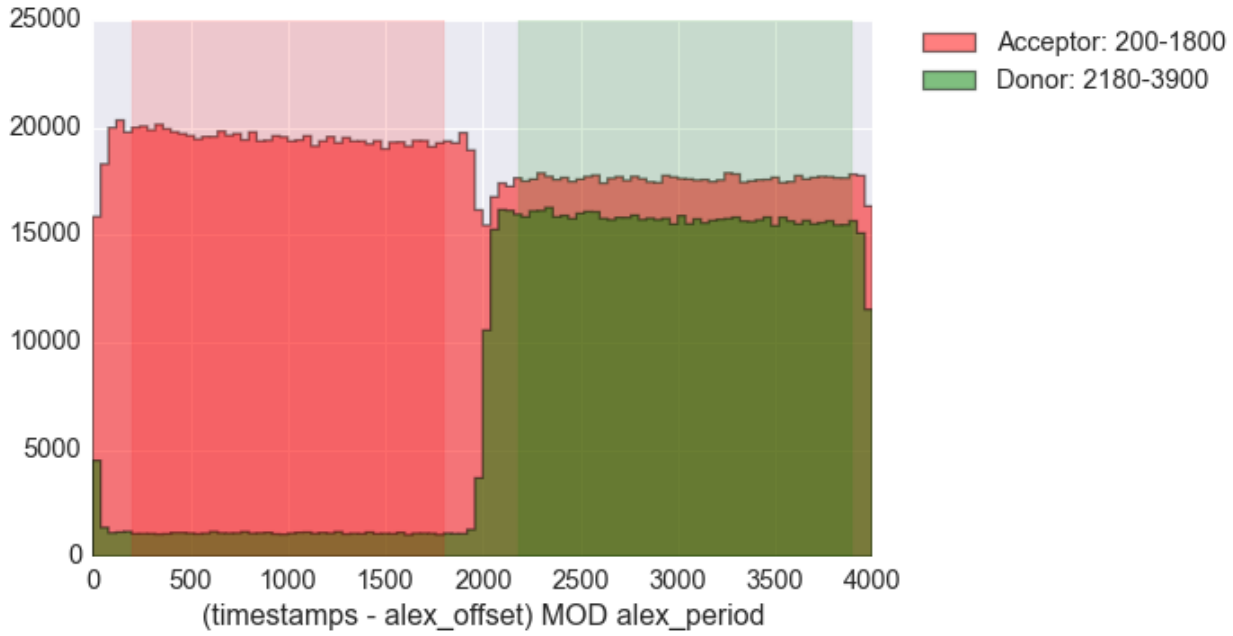


Fig. 1.1: Alternation histogram showing selection for the donor and acceptor periods.

Nanotimes time direction

In typical TCSPC measurement the *start* and *stop* inputs are inverted, i.e. the *start* is triggered by the photon and the *stop* by the the laser sync. This allows to start TAC or TDC measurements only when a photon is detected and not after each laser sync pulse. However, due to this experimental configuration, the resulting raw TCSPC histogram looks inverted along the time axis, with the nanotimes of photons emitted shortly after a laser pulse being larger than the nanotimes of photons emitted much later.

By convention, the Photon-HDF5 format requires nanotimes to be properly oriented. In other words, when a *nanotimes* time axis inversion is needed, this correction needs to be performed before the data is saved into a Photon-HDF5 file. As a corollary, TCSPC histograms computed directly from *nanotimes* from Photon-HDF5 files are always properly oriented, regardless of the way the nanotimes were acquired.

Multi-spot measurements

Multi-spot measurements are simply handled by having multiple `photon_data` groups, one for each excitation spot. The naming convention is the following:

```
photon_data0
photon_data1
...
photon_data10
...
photon_data100
```

Note that the enumeration starts from zero and there is no zero filling. Each `photon_dataN` group will have a complete `measurement_specs` sub-group so that it can effectively treated as a single-spot measurements when reading the file. As a result, even if the `measurement_type` field is not expected to change for different spots, it will be replicated inside each `photon_dataN` group.

Reading Photon-HDF5 files

You can easily read a Photon-HDF5 file in Python, MATLAB, LabVIEW or any other language that supports HDF5 (C, C++, Java, R, etc...). Photon-HDF5 files, like any other HDF5 file, are read using the HDF5 library for the language of choice. One specific advantage is that all field names (and their meaning) are defined in the specifications (*Photon-HDF5 format definition*).

To read Photon-HDF5 in a given programming language, the user only needs to install the HDF5 library and a wrapper for that language. Scientific Python distributions and MATLAB already include all the needed software support. In the case of Python, both pytables or h5py can be used. In the case of MATLAB, we suggest using 2013a or later, which include more user-friendly functions to access HDF5 files. LabVIEW users need to install the HDF5 library (www.hdfgroup.org) and a third-party wrapper to support HDF5 file reading or writing. h5labview is currently our recommended HDF5 wrapper for LabVIEW.

Simple examples on reading Photon-HDF5 files can be found in the paper describing Photon-HDF5 (section SM.2). Complete code example on reading Photon-HDF5 files in different programming languages (currently Python, MATLAB and LabVIEW) are provided on GitHub.

In the next section we discuss how to implement a generic Photon-HDF5 reader for a single-molecule FRET analysis program.

Reading Photon-HDF5 in a smFRET analysis program

This section describes an example of how to add support for reading Photon-HDF5 files to a smFRET analysis program. This scheme is implemented in the burst analysis program FRETBursts (see code for full details).

1. Get the Photon-HDF5 version from `format_version` root-node attribute (see *Root-level parameters*). The version must be `'0.4'` or greater.
2. Optionally, load metadata from *setup*, *sample*, *provenance* and *identity* groups. This is not needed for the analysis. These fields (i.e. the groups) may not be present. Even though complete Photon-HDF5 files will contain these groups, it is advisable that you write reader programs to be robust against missing metadata groups. For example detector dark-counts files we generated don't have a setup group as it it irrelevant in this situation.
3. Do the same as in point 2 for root fields *description* and *acquisition_duration*. These two fields should always be present though.
4. If there is a `/photon_data` group the file is single-spot. Call the function to load single-spot data (see next section). If there is no `/photon_data` the file is a multi-spot one, call the multi-spot read function (next section).

Read single-spot function

Make a function that takes a channel parameter (*N*) and reads the data of the corresponding channel (`/photon_dataN`). If *N* is not passed, the function reads data from `/photon_data`.

Read the photon-data following these steps:

1. Check if a photon-data group with given *N* is present, if not skip the channel. This is needed in cases of missing channels (e.g. dead pixel in SPAD array).
2. Load photon-data arrays. Load the array `timestamps` and its unit (`timestamps_specs/timestamps_unit`). If present, load also `detectors`, `nanotimes` and `particles`. If `nanotimes` is present load `nanotime_specs` (`tcspc_unit` and `tcspc_num_bins`).

3. Load measurements-specs. Refer to *Measurement specs* documentation for details. Note that `measurement_specs` may not be present.
4. If `measurement_specs` is present and the measurement-type starts with `smFRET` load detectors definitions (donor: `measurement_specs/detectors_specs/spectral_ch1`, acceptor: `measurement_specs/detectors_specs/spectral_ch2`).
5. For μs-ALEX load `alex_period`, and for ns-ALEX load `laser_repetition_rate`.
6. For both μs-ALEX and ns-ALEX, load the donor and acceptor period definitions (`alex_excitation_period1` and `alex_excitation_period1`). For μs-ALEX also load `alex_offset`. All these field may not be present.

Read multi-spot function

Implement the single-channel version then:

- Find all the root groups starting with `photon_data` and for each group load the data for that channel. There can be missing channels (e.g. if there are dead pixels).

Writing Photon-HDF5 files

To create Photon-HDF5 files, users can convert existing files or save directly from suitably modified acquisition software. The conversion option is generally the simplest approach and, when using closed-source acquisition software, also the only one available (until vendors start supporting Photon-HDF5).

To simplify saving (and converting) Photon-HDF5 files we developed and maintain, phconvert, an open-source python library serving as reference implementation for the Photon-HDF5 format. While Photon-HDF5 *can be created without phconvert*, using only a HDF5 library, we recommend taking advantage of phconvert to simplify the writing step and to make sure that the saved file conforms to the specifications. Phconvert, in fact, checks that all mandatory fields are present and have correct names and types, and adds a description to each field. Phconvert can be directly used in programs written in Python or other languages that allow calling Python code (see next sections). phconvert permissive license (MIT) allows integration with both open and closed source software.

Converting files to Photon-HDF5

phconvert includes a browser-based interface using Jupyter Notebooks to convert vendor-specific file formats into Photon-HDF5 without requiring any python knowledge. The formats currently supported are HT3 (from PicoQuant TCSPC hardware), SPC/SET (from Becker & Hickl TCSPC hardware) as well as SM (a legacy file format developed by the WeissLab, UCLA).

We provide a demo service to run these notebooks online and convert one of these formats to Photon-HDF5 without software installation on the user's computer.

Beyond the currently supported ones, other formats can be converted by writing a Python function to load the data and by using phconvert to save the data to Photon-HDF5. Taking the existing phconvert loader functions as examples, this task is relatively easy even for inexperienced Python programmers. See also the notebook Writing Photon-HDF5 files (view online).

We encourage interested users to contribute to load functions to phconvert so that out-of-the-box support for conversion of the largest number of formats can be provided. If you have an input file format not supported by phconvert please open a new issue on GitHub.

Save Photon-HDF5 from a third party-software

To directly save Photon-HDF5 files from within an acquisition software, there are several options. For programs written in Python, the obvious option is using phconvert which makes simple creating Photon-HDF5 files while assuring the validity of the output file. See for example the notebook Writing Photon-HDF5 files (view online).

For acquisition software written in other languages(e.g. C, MATLAB or LabVIEW), it is in principle possible to call python using the Python C API (see Embedding Python in Another Application). However understanding the Python C API requires a fairly good proficiency in C (and probably python).

In order to make it easy to create valid Photon-HDF5 in any language (without duplicating the effort of creating a library like phconvert in every language) we devised an alternative approach. The user can save the *photon-data arrays* (timestamps, detectors, nanotimes, etc. . .) in a plain HDF5 file. The remaining metadata is written in a simple text file (YAML). Next, a script called phforge reads the metadata and the photon-data arrays and creates a valid Photon-HDF5 file using phconvert. In this way, at the cost of a small inefficiency (writing some temporary files), a user can easily and reliably generate Photon-HDF5 files from any language. The metadata file is a text-based representation of the full Photon-HDF5 structure, excluding the photon-data arrays and some other field automatically filled by phconvert. To store this metadata we use YAML markup (a superset of JSON) for its simplicity and ability to describe hierarchical structures. For example, a minimal metadata file describing only mandatory fields is the following:

```
description: This is a dummy dataset which mimics smFRET data.

setup:
    num_pixels: 2                # using 2 detectors
    num_spots: 1                 # a single confocal excitation
    num_spectral_ch: 2           # donor and acceptor detection
    num_polarization_ch: 1       # no polarization selection
    num_split_ch: 1              # no beam splitter
    modulated_excitation: False  # CW excitation, no modulation
    lifetime: False              # no TCSPC in detection

photon_data:
    timestamps_specs:
        timestamps_unit: 10e-9   # 10 ns
```

To save the photon-data arrays the user needs to call the HDF5 library for the language of choice. For example, in MATLAB timestamps and detectors arrays can be saved with the following commands:

```
h5create('photon_data.h5', '/timestamps', size(timestamps), 'Datatype', 'int64')
h5write('photon_data.h5', '/timestamps', timestamps)
h5create('photon_data.h5', '/detectors', size(detectors), 'Datatype', 'uint8')
h5write('photon_data.h5', '/detectors', detectors)
```

Finally, once metadata and photon-data files have been saved, a Photon-HDF5 file can be created calling the phforge script as follows:

```
phforge metadata.yaml photon-data-arrays.h5 photon-hdf5-output.hdf5
```

Note that the file generated with this minimal metadata, does not contain the *measurement_specs group* which is in general necessary for a user to analyze the data.

The phforge script is available at http://photon-hdf5.github.io/phforge/. Examples of complete metadata files for all the supported measurement types are available at https://github.com/Photon-HDF5/phforge/tree/master/example_data.

A complete example of creating Photon-HDF5 files in LabVIEW using phforge can be found at https://github.com/Photon-HDF5/photon-hdf5-labview-write (for a MATLAB see next section).

Please use the mailing list if you have any questions.

Saving Photon-HDF5 from MATLAB

Creating Photon-HDF5 in MATLAB is easy using the approach described in the previous section, i.e. calling the script phforge.

Complete MATLAB examples can be found at https://github.com/Photon-HDF5/photon-hdf5-matlab-write.

In principle, it should be possible using a recent release of MATLAB (R2014b or later) to directly call python functions. Therefore it should be possible to directly call phconvert. However, in our recent attempt, we weren't able to configure MATLAB in order to load the correct dynamic libraries (i.e. the HDF5 C library) required by phconvert.

Saving Photon-HDF5 from scratch using only an HDF5 library

To create Photon-HDF5 files from languages different than python the easiest option, by far, is calling the phforge script as described in previous section *Save Photon-HDF5 from a third party-software*.

If for some reason you cannot use phforge or phconvert, you have to implement routines to write Photon-HDF5 files using the HDF5 library for your platform, taking care of following the Photon-HDF5 specification. In the following paragraph we provide a few suggestions on how to proceed in this case.

To facilitate writing valid Photon-HDF5, we provide a JSON file containing all the official field names, a short description and a generic type definition (array, scalar, string or group). This JSON file can be used both to validate names and types of the data fields and to retrieve the standard short description (this is, in fact, what *phconvert* does). The developer needs to verify that all the mandatory fields are present. The description string should be saved for all the official fields in an attribute named "TITLE". For compatibility with h5labview, we recommend to use a single-space string (" ") for all the user fields that lack a description (phconvert uses this workaround too).

Furthermore, the */identity group* should include the fields `software_name` and `software_version` to specify the name and the version of the software that created the file.

Finally, you can verify that generate files are compliant with the Photon-HDF5 specifications by using the *phconvert* function phconvert.hdf5.assert_valid_photon_hdf5_tables(). This function will raise errors or warnings if the input file does not follows the specs.

Defining new measurement types

If you need to save a type of measurement not included in the list of currently supported measurement types, you need to define a new "measurement_specs", i.e. which info to put in the *measurement_specs group* in order to make the dataset self-contained.

What to put in measurement_specs

First and foremost, a new measurement_specs needs to have a name, i.e. the name associated with the type of measurement to be saved. The name of the measurement-type is a string stored in *measurement_specs/measurement_type*. This string is used to differentiate between different measurement_specs. For example, names of already defined measurment types are: *smFRET*, *smFRET-usALEX*, *smFRET-usALEX-3c* and *smFRET-nsALEX* (see *here*).

Beyond the name, measurement_specs group should contain all the metadata needed to for unambiguous analysis of the dataset. For example, in smFRET experiments the measurement_specs contains the IDs of donor and acceptor detection channels. In Alternated Excitation (ALEX) measurements it will also contains definition of laser alternation (period, range for donor and acceptor, etc...).

In most cases, measurement_specs will contain the association between detector IDs physical detection channels (*Detectors specs*). Detection channels may differ in the detected spectral band and in detected polarization, called

spectral_chX and *polarization_chX* (where *X* is 1, 2, ...). When using a non-polarizing beam splitter and both spectral band and polarization are equal, the detection channels are names *split_chX* (see *Beam-split channels*).

If possible, new measurement_specs should follow naming conventions of other official measurement_specs.

For any question please use the Photon-HDF5 Google Group.

How to propose a new measurement_specs

Logistically, the process works as follows.

1. Users propose new *measurement_specs* using the Photon-HDF5 Google Group.
2. After discussion, the measurement_specs will be advertised on the Photon-HDF5 website in order to gather opinions from different parties.
3. Discussion continues and modifications are proposed until consensus is reached.
4. Finally, the new measurement_specs becomes official part of Photon-HDF5: it is added to the Photon-HDF5 documentation and software support is added to phconvert.

For any question please use the Photon-HDF5 Google Group.

Known Limitations

In this section, we list some features that are not currently supported. If you think that some of these should be included in the specifications, please contact us.

Timestamps with rollover

In Photon-HDF5 timestamps are always signed 64 bit integers. Thanks to compression, there is no size penalty compared to 32 bit integers. Most timestamping hardware produce a timestamp with 24 or 32 bits and a rollover flag in order to compute the full "unwrapped" timestamp. Saving timestamps with a separate rollover information is not currently supported, therefore the rollover correction must be computed before saving data in a Photon-HDF5 file.

Timestamps with rollover may be supported in a future version of Photon-HDF5.

Collaborating

The success of a file format is only determined by the extent of its adoption. For this reason we greatly welcome any feedback and contribution from interested users.

How to participate?

The Photon-HDF5 project resources include:

1. Reference Documentation (i.e. this document).
2. Examples on reading Photon-HDF5 in multiple languages.
3. Software for saving/converting Photon-HDF5 files: phconvert (reference python library); phforge (script to be used from any language); Photon-HDF5 Online Converter (demostrative service).

All the sources (including for the documentation) are hosted on GitHub and we encourage to open *GitHub Issues* in the documentation repository to discuss any topic related Photon-HDF5. You can also contact us by email, but we prefer to use GitHub in order to keep any discussion public.

Contributions (such as fixes or enhancements) can be sent using *GitHub Pull Requests* (PR). You can find guides on how to send a PR on the GitHub website. If have have any doubts, please contact us on the Photon-HDF5 google group and we will be glad to help you getting started.

There are several ways you can get involved:

- **Sending feedback**: if you use or plan to use Photon-HDF5 and have any comment or suggestion, please send it to us! Even if you don't have any problem we would like to hear back about your use case. For this topic please use the Photon-HDF5 google group or open an issue in the documentation repository.
- **Documentation contributions**: if you feel that some section of this document should be expanded or enhanced in any way, please feel free to open an issue or send a PR (see note above) on the documentation repository.
- **Contributing examples**: you can send a new example on reading Photon-HDF5 files in a new language or for a different measurement type. Or simply send a fix for the current examples.
- **Contributing to phconvert**: you can open issues to report bugs, discuss usage or propose enhancements. You are also more than welcome to send PR for fixes or enhancements to the library. The official repository is this one.

Contributions Acknowledgement

Any contributions to this documentation will be listed in the front page, just below the authors.

Contributions to other repository (e.g. phconvert, phforge, reading examples, etc.) will be acknowledged in the respective website and listed in their `LICENSE` files. All the code in Photon-HDF5 projects is released under the MIT license.

Contributor Code of Conduct

The Photon-HDF5 team subscribes to the Contributor Covenant, version 1.0.0, available from http://contributor-covenant.org/version/1/0/0/.