SUPPORTING MATERIAL

Photon-HDF5: an open file format for single-molecule fluorescence experiments using photon-counting detectors

Antonino Ingargiola[1], Ted Laurence[2], Robert Boutelle[1], Shimon Weiss[1], Xavier Michalet[1]

1 Department of Chemistry and Biochemistry, University of California Los Angeles, CA, USA.
2 Physical and Life Sciences Directorate, Lawrence Livermore National Laboratory, Livermore, CA, USA.

SM1. Photon-HDF5 files examples

In this section, we describe a few examples of Photon-HDF5 files corresponding to different types of measurement, focusing for brevity on the mandatory fields in */photon_data*. The fields here shown, as well as all the other fields in the remaining groups (*/setup*, */sample*, */identity* and */provenance*) can be found in the section "Photon-HDF5 format definition" of the reference documentation(1).

Photon-HDF5 files can be distinguished from other HDF5 files by the presence of two root node attributes *format_name* and *format_version*: the former contains the string "Photon-HDF5" and the latter is a string identifying the Photon-HDF5 version (currently "0.4"). The measurement type is identified by reading the string in */photon_data/measurement_specs/measurement_type*, whose currently supported values are reported in Table SM-1.

SM1.1 µs-ALEX

In µs-ALEX smFRET data files, the following data fields are present (their definitions can be found in the reference manual(1)):

1. Timestamps (array): */photon_data/timestamps*
2. Timestamps unit (scalar): */photon_data/timestamps_specs/timestamps_unit*
3. Detectors array (array): */photon_data/detectors*
4. Measurement type (string): */photon_data/measurement_specs/measurement_type* (smFRET-usALEX)

Additionally, the */photon_data/measurement_specs* group contains the fields indicated in Table SM-1, row *smFRET-usALEX*.

SM1.2 ns-ALEX

In ns-ALEX smFRET (also known as PIE) data files, the following data fields are present:

1. Timestamps (array): */photon_data/timestamps*
2. Timestamps unit (scalar): */photon_data/timestamps_specs/timestamps_unit*
3. Detectors array (array): */photon_data/detectors*
4. TCSPC nanotimes (array): */photon_data/nanotimes*

5. TCSPC bin width (scalar): */photon_data/nanotimes_specs/tcspc_unit*

6. TCSPC number of bins (scalar): */photon_data/nanotimes_specs/tcspc_num_bins*

7. Measurement type (string): */photon_data/measurement_specs/measurement_type* (smFRET-nsALEX)

Additionally, the */photon_data/measurement_specs* group contains the fields indicated in Table SM-1, row *smFRET-nsALEX*. Definitions for all these fields can be found in the "Photon-HDF5 format definition" section of the reference manual(1).

Table SM-1: List of currently supported measurement types. Additional types can be added in the future based on user demand. The first column represents the string identifying the measurement type that is located at */photon_data/measurement_specs/measurement_type* in the Photon-HDF5 file. The second column is a brief description of the measurement type. The third column lists the required metadata fields in */photon_data/measurement_specs/* for each type of measurement.

Measurement type	**Description**	**Required fields in *measurement_specs***
smFRET	2-colors smFRET with single wavelength excitation	• *detectors_specs/spectral_ch1* • *detectors_specs/spectral_ch2*
smFRET-usALEX	2-colors smFRET with 2 alternating CW excitation lasers	• *detectors_specs/spectral_ch1* • *detectors_specs/spectral_ch2* • *alex_period* • *alex_offset* • *alex_excitation_period1 (optional)* • *alex_excitation_period2 (optional)*
smFRET-usALEX-3c	3-colors smFRET with 3 alternating CW excitation lasers	• *detectors_specs/spectral_ch1* • *detectors_specs/spectral_ch2* • *detectors_specs/spectral_ch3* • *alex_period* • *alex_offset* • *alex_excitation_period1* • *alex_excitation_period2* • *alex_excitation_period3*
smFRET-nsALEX	2-colors TCSPC-based smFRET with 2 alternating pulsed excitation lasers (also known as PIE)	• *detectors_specs/spectral_ch1* • *detectors_specs/spectral_ch2* • *laser_repetition_rate* • *alex_excitation_period1 (optional)* • *alex_excitation_period2 (optional)*

Figure SM-1: Partial diagram description of the Photon-HDF5 layout comprising the *photon_data* and *setup* groups, which contains the acquisition data and all the information needed for data processing. The symbol preceding each field indicates the data type of that field (e.g. array, string, scalar or boolean), as described in the legend.

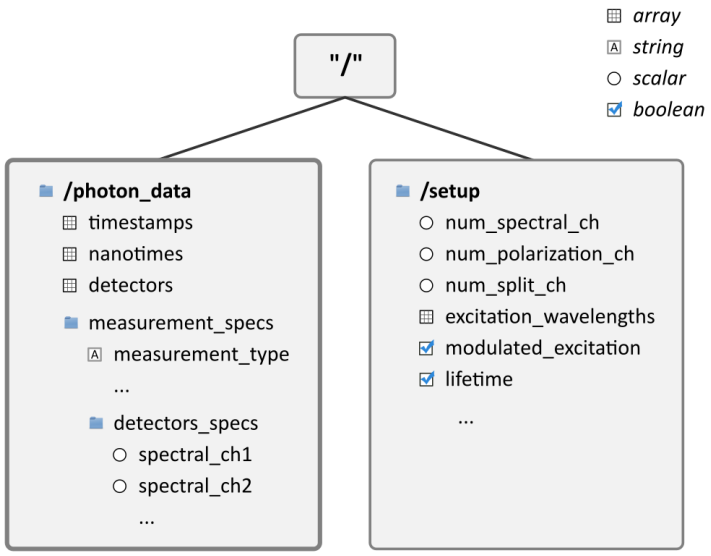
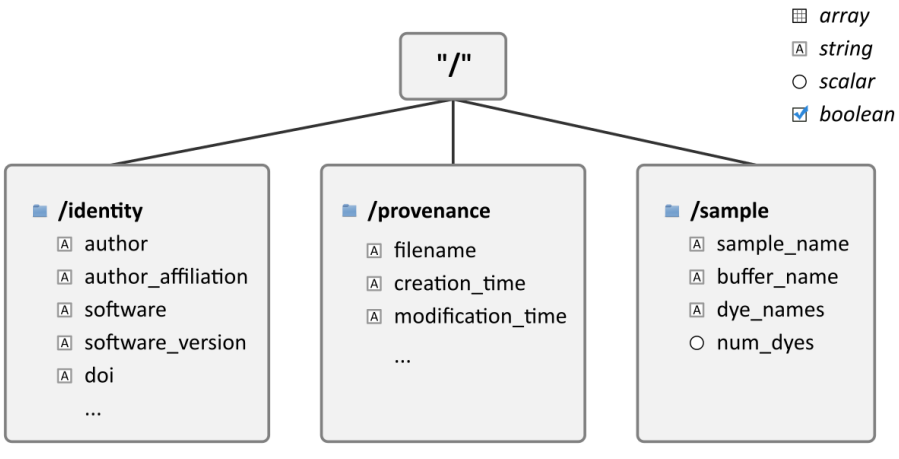


Figure SM-2: Partial diagram description of the Photon-HDF5 layout showing the *sample*, *identity* and *provenance* groups. These groups are not strictly necessary to analyze the data, but are important for long-term preservation, reproducibility and proper credit/citation when the file is shared.



SM2. Reading Photon-HDF5 Files

In this section, we show basic examples of how to read a Photon-HDF5 file containing µs-ALEX data in Python (SM2.1), MATLAB (SM2.2) and LabVIEW (SM2.3). These and other examples of reading Photon-HDF5 files are provided online (2). Additional information on reading Photon-HDF5 files can be found in the section "Reading Photon-HDF5 files" of the reference documentation(3).

SM2.1 Reading in Python

Using Python and the pytables, we start by opening the data file and getting a handle to the photon-data group:

```
h5file = tables.open_file(filename)
photon_data = h5file.root.photon_data
```

Next, to read the timestamps and detectors array and the timestamps units, we use:

```
timestamps = photon_data.timestamps.read()
timestamps_unit = photon_data.timestamps_specs.timestamps_unit.read()
detectors = photon_data.detectors.read()
```

To retrieve donor and acceptor detector ID and other µs-ALEX specific fields:

```
donor_ch = photon_data.measurement_specs.detectors_specs.spectral_ch1.read()
acceptor_ch = photon_data.measurement_specs.detectors_specs.spectral_ch2.read()
alex_period = photon_data.measurement_specs.alex_period.read()
offset = photon_data.measurement_specs.alex_offset.read()
donor_period = photon_data.measurement_specs.alex_excitation_period1.read()
acceptor_period = photon_data.measurement_specs.alex_excitation_period2.read()
```

Finally, some summary information can be printed as follow:

```
print('Number of photons: %d' % timestamps.size)
print('Timestamps unit:   %.2e seconds' % timestamps_unit)
print('Detectors:         %s' % np.unique(detectors))
print('Donor CH: %d     Acceptor CH: %d' % (donor_ch, acceptor_ch))
print('ALEX period: %4d \nOffset: %4d \nDonor period: %s \nAcceptor period: %s' %
      (alex_period, offset, donor_period, acceptor_period))
```

This and other Python examples (using both pytables and h5py) can be found at:

https://github.com/Photon-HDF5/photon_hdf5_reading_examples/tree/master/python

SM2.2 Reading in MATLAB

The following example works in MATLAB 2013a or later. For earlier versions of MATLAB (7.3 or later), the user may use the low-level HDF5 functions (which are however more complex).

To load timestamps and detectors arrays and timestamps unit, we execute:

```
timestamps = h5read(filename, '/photon_data/timestamps');
timestamps_unit = h5read(filename, '/photon_data/timestamps_specs/timestamps_unit');
detectors = h5read(filename, '/photon_data/detectors');
```

Next, to retrieve donor and acceptor detector ID and other µs-ALEX specific fields:

```
donor_ch = h5read(filename, '/photon_data/measurement_specs/detectors_specs/spectral_ch1');
acceptor_ch = h5read(filename, '/photon_data/measurement_specs/detectors_specs/spectral_ch2');
alex_period = h5read(filename, '/photon_data/measurement_specs/alex_period');
offset = h5read(filename, '/photon_data/measurement_specs/alex_offset');
donor_period = h5read(filename, '/photon_data/measurement_specs/alex_excitation_period1');
acceptor_period = h5read(filename, '/photon_data/measurement_specs/alex_excitation_period2');
```

And to print some summary information:

```
fprintf('Number of photons: %d\n', size(timestamps));
fprintf('Timestamps unit:   %.2e seconds\n', timestamps_unit);
fprintf('Detectors:         %s\n', unique(detectors));
fprintf('Donor CH: %d     Acceptor CH: %d\n', [donor_ch; acceptor_ch]);
fprintf('ALEX period: %d \nOffset: %d \nDonor period: %d , %d      Acceptor period: %d , %d\n',...
        [alex_period; offset; donor_period; acceptor_period]);
```

The full example can be found at:

https://github.com/Photon-HDF5/photon_hdf5_reading_examples/tree/master/matlab

SM2.3 Reading in LabVIEW

Figure SM-3 shows a snapshot of one of the two LabVIEW codes provided as examples at

https://github.com/Photon-HDF5/photon_hdf5_reading_examples/tree/master/labview

The code is saved in LabVIEW 2010 and should be compatible with later versions of LabVIEW. The code uses the h5labview wrapper for the HDF5 library. This requires a two-step installation. First, installation of the HDF5 library available at https://www.hdfgroup.org/ftp/HDF5/releases/hdf5-1.8.14/bin/windows/. At the time of this writing, the current tested version of the library is 1.8.14. Second, installation of h5labview (available at http://sourceforge.net/projects/h5labview/files/) using VPI Manager, a free utility to install LabVIEW packages. The examples have been generated using version 2.11.2.137 of h5labview2, but should work with later versions as well.

The only user input is the path of the file to be read (Path parameter on the front panel - not shown on Fig. SM-3). Pressing the “Run” button will load the file and display some basic information about its content, the raw data (timestamp and detector arrays) as well as represent the time stamp histogram modulo the alternation period and the donor and acceptor emission periods as they were defined when saving the file.

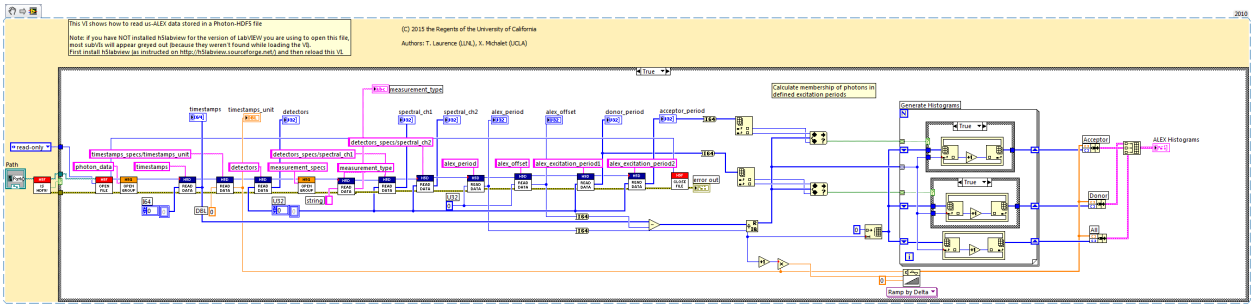


Figure SM-3: Example LabVIEW 2010 code to read a Photon-HDF5 file corresponding to a µs-ALEX measurement.

SM3. Writing Photon-HDF5 Files

In the following sections, we continue the discussion of section 3.3.3 in the main text, on converting or saving Photon-HDF5 files.

As mentioned before, supported file formats can be directly converted using one of the Jupyter notebooks included in phconvert. For unsupported file formats, a user can write a Python function to load the data, and use *phconvert* to save it into the Photon-HDF5 file.

Users may want to implement the ability to save Photon-HDF5 files in software for data-acquisition or simulators. For Python programs the direct solution is calling phconvert to save the file as illustrated in SM-3.1. As mentioned before, using phconvert greatly simplifies writing Photon-HDF5 files while ensuring compatibility with the Photon-HDF5 specifications. For non-Python programs which cannot easily call phconvert, saving Photon-HDF5 files is equally easy (and reliable) using the phforge script(4) as described in SM-3.2. Under the hood, phforge calls the phconvert library and therefore creates files which are guaranteed to be valid Photon-HDF5 files.

SM3.1. Saving Photon-HDF5 from Python programs: phconvert

In this section, we provide an example of how to write a Photon-HDF5 in Python with phconvert. This example illustrates how to build the data structure needed to save a Photon-HDF5 file. This example can be also used as a reference by users willing to implement a loader function for an additional format.

The logic is the following. Each group in a Photon-HDF5 file is represented in memory by a Python dictionary: a key-value pair (in the dictionary) represents a field name and its content respectively (in the Photon-HDF5 file). For example, the root group dictionary contains an item whose key is `'description'` (i.e. the HDF5 field name) and whose value is a string (i.e. the HDF5 field content).

Following this logic, the two mandatory groups of a Photon-HDF5 file (*photon_data* and *setup*) can be defined as follows:

```
photon_data = dict(
    timestamps=timestamps,  # array of int64
    detectors=detectors,    # array of uint8
    timestamps_specs={'timestamps_unit': 10e-9})  # 10 ns

setup = dict(
    ## Mandatory fields
    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

    ## Optional fields
    excitation_wavelengths = [532e-9],         # List of excitation wavelengths
    excitation_cw = [True],                    # List of booleans, True if wavelength is CW
    detection_wavelengths = [580e-9, 640e-9],  # Nominal center wavelength for each detection ch
)
```

Note that a subgroup (such as *timestamps_specs* in *photon_data*) is simply a nested dictionary. We need to merge this data to create the root group of the Photon-HDF5 file, which also requires a *description* field:

```
data = dict(
    description = 'This is a dummy dataset which mimics smFRET data.',
    photon_data = photon_data,
    setup = setup)
```

Finally, the data can be saved to disk by calling the function *save_photon_hdf5* in phconvert:

```
import phconvert as phc
phc.hdf5.save_photon_hdf5(data, h5_fname='dummy_dataset.h5')
```

This will save a minimal Photon-HDF5 file. Phconvert will add a few additional fields that can be generated automatically (e.g. *acquisition_duration*, field *software* in the *identity* group, etc), a description for each field and will validate the field names and types to conform to the specifications. This file will not include a *measurement_specs* group. A complete example using non-mandatory fields (including the *measurement_specs* group) is included in phconvert and accessible at:

http://nbviewer.ipython.org/github/Photon-HDF5/phconvert/blob/master/notebooks/Writing Photon-HDF5 files.ipynb

SM3.2. Saving Photon-HDF5 from non-Python programs: phforge

Users may want to implement the ability to save Photon-HDF5 files in software for data-acquisition or simulators not written in Python. As mentioned before, using phconvert would greatly simplify writing Photon-HDF5 files while ensuring compatibility with the Photon-HDF5 specifications. However, calling phconvert from other languages can be problematic. One approach is calling the Python C API to call phconvert, but this is not a particularly easy task. Alternatively, users could write Photon-HDF5 files directly using the HDF5 library for the language of choice. However, in order to ensure the creation of valid Photon-HDF5 files, the program should check the spelling of all the field names, their data types, add the official field descriptions and make sure that all the mandatory fields are presents. Most of these validations can be implemented using the Photon-HDF5 JSON specifications file. In practice, one would need to re-implement the same functionalities of phconvert, but in another language.

In order to make it easy to create validated Photon-HDF5 in any language and bypass this reimplementation step, we devised an alternative approach in which the Photon-HDF5 is created using a script called phforge(4). The process involves three simple steps:

1. Save the mandatory photon-data arrays (timestamps, detectors, nanotimes, etc…) in a plain HDF5 file.
2. Write the remaining metadata in a simple text file (in YAML format, see below).
3. Call *phforge* providing metadata and photon-data file names as input arguments to create a valid Photon-HDF5 file using phconvert.

These three steps are now briefly discussed.

1. To save the photon-data arrays, the user needs to call the HDF5 library for the language he or she chooses to use. In MATLAB for example, *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)
```

A similar example using LabVIEW is provided at https://github.com/Photon-HDF5/photon-hdf5-labview-write.

2. The metadata file is a text-based representation of the full Photon-HDF5 structure, excluding the photon-data arrays and a few other fields 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 containing only mandatory fields looks as follows:

```
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
```

3. 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
```

where *metadata.yaml* designates the path to the metadata file, *photon-data-array.h5* the path to the temporary HDF5 file (containing only photon-data arrays) and *photon-hdf5-output.hdf5*, the path of the destination file.

Note that the file generated with this minimal metadata example will not contain a *measurement_specs* group, which is in general necessary for a user to analyze the data. The fields corresponding to the measurement_specs group (or any other valid Photon-HDF5 field) need to be added to the metadata file, so that phforge can incorporate them in the final photon-hdf5 file.

The phforge script is available online(4) and is easy to install on all the major operating systems. More examples of metadata files, including non-mandatory fields and measurement_specs group for different types of measurements, are available at:

https://github.com/Photon-HDF5/phforge/tree/master/example_data

A complete example of how to create Photon-HDF5 files in MATLAB is provided at:

https://github.com/Photon-HDF5/photon-hdf5-matlab-write

A complete example of how to create Photon-HDF5 files in LabVIEW is provided at:

https://github.com/Photon-HDF5/photon-hdf5-labview-write

No matter which language the acquisition software is written in, it should be remembered that Photon-HDF5 files need, in most cases, some pre-processing of the data. For example, timestamps and detectors arrays needs to be separated (they are often packed in a common structure) and overflow corrections needs to be applied to timestamps (which are usually recorded with 32 or less bits by the acquisition hardware). Additionally, for µs-ALEX or ns-ALEX experiments, information on the alternation periods should be provided (although it is not mandatory). This processing can be performed either in real-time during the acquisition or in a second post-acquisition step. The latter approach naturally fits with using the phforge script for creating the final Photon-HDF5 file.

As a final note, if the developer decides to modify his or her acquisition software to save Photon-HDF5 files, he or she should also consider providing the user a simple way to input the required metadata. When converting files with a phconvert notebook, this metadata is entered using the notebook interface. When using the phforge script in custom acquisition software this metadata can be acquired by manually editing a pre-formatted YAML file or through a custom GUI which generates the metadata file automatically. The latter approach, for example, is followed in the previously linked LabVIEW example for writing Photon-HDF5 files.

SM4. Benchmarks

SM4.1 Overview

In this section, we report the results of two simple read- and write-speed benchmarks for two different data files: one based on a HT3 file (created by the MicroTime 200 hardware manufactured by PicoQuant) and one based on a SPC file (created by a SPC-630 board manufactured by Becker & Hickl). The benchmarks were performed in Python 2.7 and pytables 3.1 and run on a Windows 7 x64 desktop PC with Intel® Core™ i5-3570K CPU, 16GB of RAM and a 7,200 rpm SATA hard drive. The quoted values are the result of a single execution after a system reboot (to eliminate perturbation from file-system cache).

It should be noted that the value reported are only indicative and cannot be generalized because compression performances (size and speed) can significantly vary from file to file. Furthermore HDF5 allows changing one parameter, the array chunk size (the size of an on-disk-contiguous chunk of data), which can affect (in particular, improve) performance. Here, we used the default value as set by the pytables library.

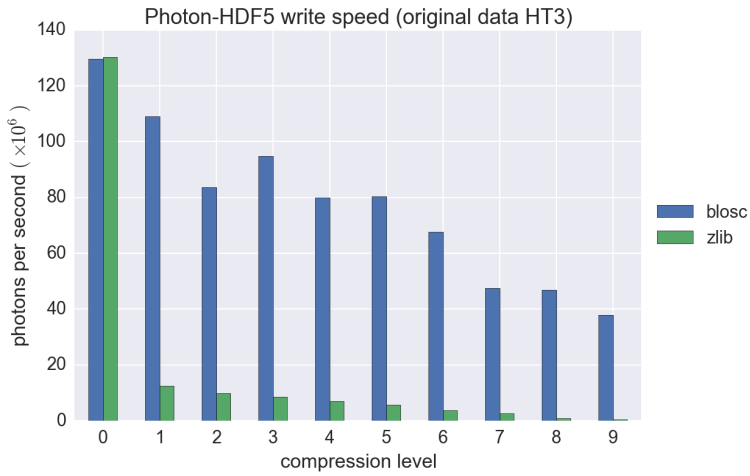
We benchmarked different compression levels ranging from 0 (no compression) to 9 (maximum compression) and 2 compression algorithms: zlib (HDF5 default) and blosc (a high-speed alternative compressor developed by the pytables team and not shipped by default with the HDF5 library). It is important to note that, when using languages other than Python, using blosc requires compiling the blosc sources, which can represents a significant obstacle for the user. Therefore, we discourage the use of the blosc compressor for any file which needs to be shared. On the other hand, we included it in the benchmarks due to its impressive performance. For instance, in critical, real-time applications where the hard drive write-speed is a bottleneck (i.e. acquiring from large SPAD arrays), the blosc compressor could be used and the conversion to zlib postponed to a second, post-processing step.

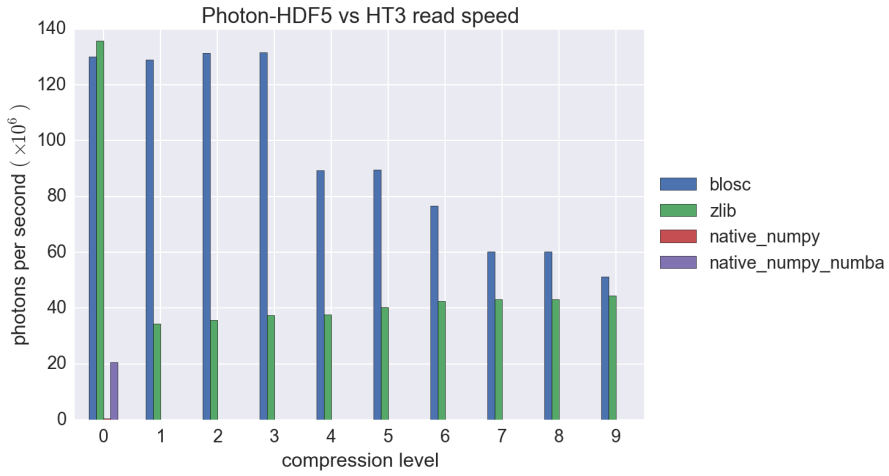
SM4.2 Results

Read speeds were relatively uniform at all compression levels when using zlib, with values around 40 million photons per second (MP/s). In all examples, the read speed from compressed Photon-HDF5 format (compression level of 5) was around twice faster than reading from the native formats (the latter have an overhead due to byte decoding and rollover correction). It is worth noting that the read speed of the native formats depends on the specific decoding implementation, and C routines could potentially result in faster read speed (although we believe our implementation is reasonably fast).

Write speeds, on the contrary, were strongly affected by the compression level (5-10 MP/s using zlib5). Obviously, the fastest was to write uncompressed data (> 100 MP/s). Remarkably, blosc exhibited write speeds which were comparable to the uncompressed one, while significantly reducing the file size. For single-spot excitation data files, the write speed was more than enough to save data in real time even when using the slow zlib compression. For setup using large detector arrays, it would be advisable to not use any compression or only blosc compression (and instead convert files with zlib compression post acquisition or for sharing).

Figure SM-4: Write and Read speed benchmark using a PicoQuant HT3 input file. In the first two graphs (Write and Read speed, respectively), larger values are better, while in the last (size), smaller is better. Compression level of 0 indicates no compression.





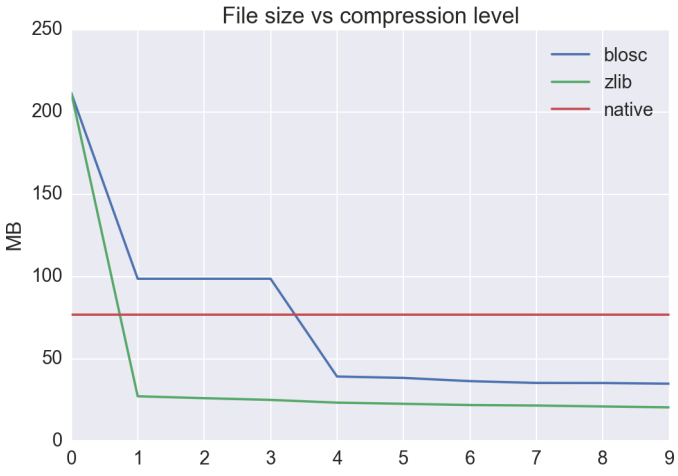
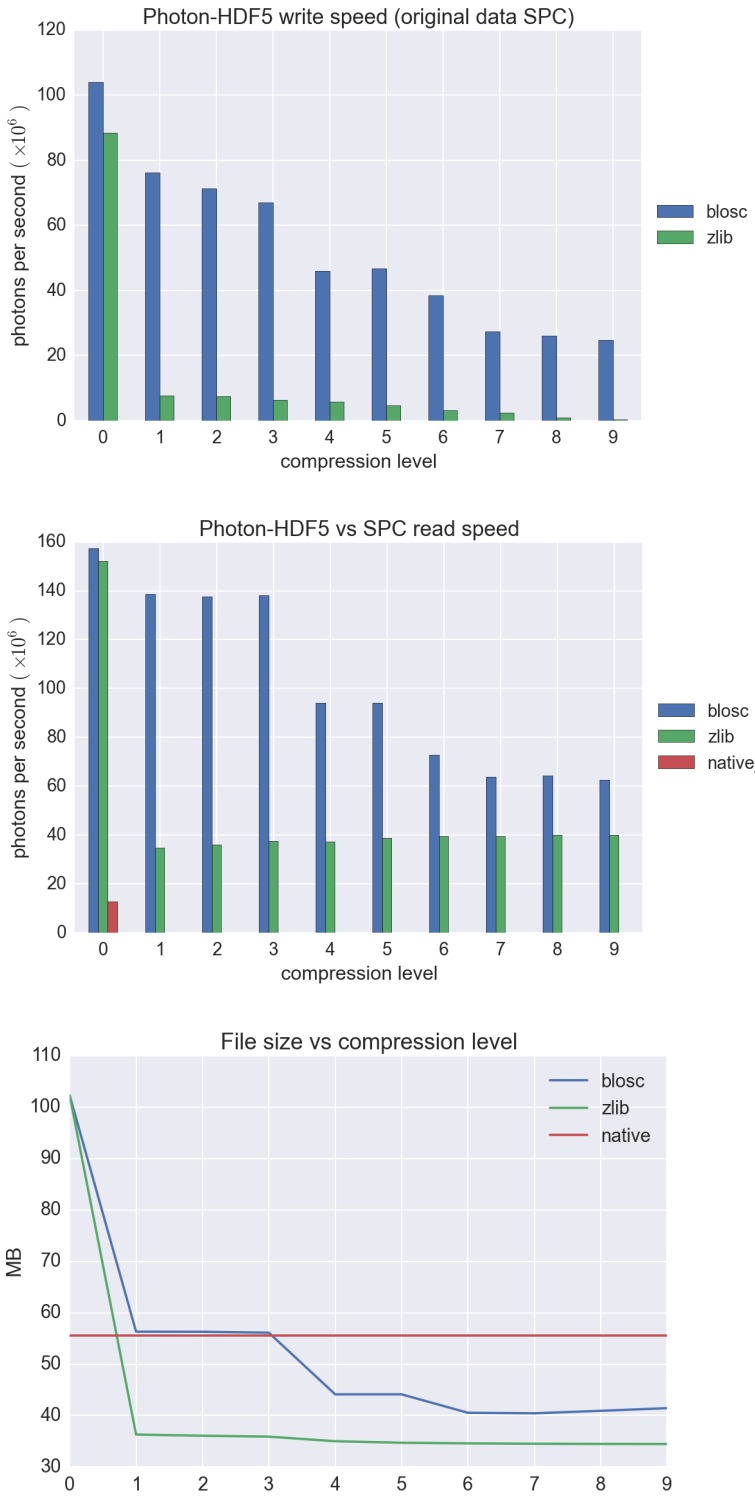


Figure SM-5: Write and Read speed benchmark using a Becker & Hickl SPC input file. In the first two graphs (Write and Read speed, respectively), larger values are better, while in the last (size), smaller is better. Compression level of 0 indicates no compression.



Supporting Material References

1. Photon-HDF5 format definition. http://photon-hdf5.readthedocs.org/en/latest/phdata.html.

2. Reading Photon-HDF5 in multiple languages. http://photon-hdf5.github.io/photon_hdf5_reading_examples/.

3. Photon-HDF5 Reference: Reading Photon-HDF5 files. http://photon-hdf5.readthedocs.org/en/latest/reading.html.

4. phforge: a script for creating Photon-HDF5 files. http://photon-hdf5.github.io/phforge/.