
qpformat Documentation

Release 0.8.0

Paul Müller

Mar 04, 2019

1	Introduction	3
1.1	Why qformat?	3
1.2	Supported file formats	3
2	Getting started	5
2.1	Installing qformat	5
2.2	User API	5
2.2.1	Basic Usage	5
2.3	Command-line program “qpinfo”	6
3	Examples	7
3.1	Hologram from tif file	7
3.2	HyperSpy hologram file format	9
3.3	Conversion of external file formats to .npy files	10
3.4	Conversion of external holograms to .tif files	11
4	Code reference	13
4.1	module-level	13
4.2	file format base classes	14
4.2.1	SeriesData	14
4.2.2	SingleData	15
4.3	file format readers	16
4.3.1	SeriesFolder	16
4.3.2	SeriesHdf5HyperSpy	16
4.3.3	SeriesHdf5Qpimage	17
4.3.4	SeriesHdf5QpimageSubjoined	17
4.3.5	SeriesZipTifHolo	17
4.3.6	SeriesZipTifPhasics	17
4.3.7	SingleHdf5Qpimage	17
4.3.8	SingleNpyNumpy	18
4.3.9	SingleTifHolo	18
4.3.10	SingleTifPhasics	18
4.4	exceptions	18
5	Changelog	19
5.1	version 0.8.0	19
5.2	version 0.7.1	19

5.3	version 0.7.0	19
5.4	version 0.6.4	19
5.5	version 0.6.3	19
5.6	version 0.6.2	20
5.7	version 0.6.1	20
5.8	version 0.6.0	20
5.9	version 0.5.1	20
5.10	version 0.5.0	20
5.11	version 0.4.4	20
5.12	version 0.4.3	20
5.13	version 0.4.2	20
5.14	version 0.4.1	20
5.15	version 0.4.0	21
5.16	version 0.3.5	21
5.17	version 0.3.4	21
5.18	version 0.3.3	21
5.19	version 0.3.2	21
5.20	version 0.3.1	21
5.21	version 0.3.0	21
5.22	version 0.2.1	21
5.23	version 0.2.0	22
5.24	version 0.1.6	22
5.25	version 0.1.5	22
5.26	version 0.1.4	22
5.27	version 0.1.3	22
5.28	version 0.1.2	22
5.29	version 0.1.1	23
5.30	version 0.1.0	23
6	Bibliography	25
7	Indices and tables	27
	Bibliography	29

Qpformat is a Python3 library for opening quantitative phase imaging data file formats. This is the documentation of qpformat version 0.8.0.

1.1 Why qpformat?

There is a multitude of phase-imaging techniques that inevitably comes with a broad range of quantitative phase imaging (QPI) file formats. In addition, raw data, such as digital holographic microscopy (DHM) images, must be preprocessed to access the phase encoded in the interference pattern. Qpformat provides a unified and user-friendly interface for loading QPI data. It is based on the `qpimage` library and thus benefits from its hdf5-based data structure (e.g. elaborate background correction, meta data management, and transparent data storage). Furthermore, qpformat can manage large datasets (e.g. many holograms in one folder) without running out of memory by means of its lazily-evaluated `SeriesData` class.

1.2 Supported file formats

Class	Storage type	Description
<code>SeriesFolder</code>	<i>multiple</i>	Folder-based wrapper file format
<code>SeriesHdf5Hyper</code>	<i>hologram</i>	HyperSpy hologram series (HDF5 format)
<code>SeriesHdf5Qpimage</code>	<i>phase,amplitude</i>	Qpimage series (HDF5 format)
<code>SeriesHdf5QpimageSubjoined</code>	<i>phase,amplitude</i>	Subjoined qpimage series (HDF5 format), may contain other data
<code>SeriesZipTifHolo</code>	<i>hologram</i>	Off-axis hologram series (zipped TIFF files)
<code>SeriesZipTifPhasics</code>	<i>phase,intensity</i>	Phasics series data (zipped “SID PHA*.tif” files)
<code>SingleHdf5Qpimage</code>	<i>phase,amplitude</i>	Qpimage single (HDF5 format)
<code>SingleNpyNumpy</code>	<i>multiple</i>	Numpy complex field or phase data (numpy binary format)
<code>SingleTifHolo</code>	<i>hologram</i>	Off-axis hologram image (TIFF format)
<code>SingleTifPhasics</code>	<i>phase,intensity</i>	Phasics image (“SID PHA*.tif”)

2.1 Installing qpformat

qpformat is written in pure Python and supports Python version 3.5 and higher. qpformat depends on several other scientific Python packages, including:

- `numpy`,
- `scipy`,
- `qpimage` (phase data manipulation),

To install qpformat, use one of the following methods (package dependencies will be installed automatically):

- **from PyPI:** `pip install qpformat`
- **from sources:** `pip install -e . or python setup.py install`

2.2 User API

Qpformat supports *several file formats* that are categorized into `qpformat.file_formats.SingleData` (the experimental data file format contains only one phase image) and `qpformat.file_formats.SeriesData` (the experimental data file format supports multiple phase images). From these base classes, all data file formats are derived. The idea is that experimental data is not loaded into memory until the `get_qpimage` method is called which returns a `qpimage.QPImage` object.

2.2.1 Basic Usage

To extract the (unwrapped) phase from a DHM image, use the `qpformat.load_data()` method. The file format type is determined automatically by qpformat.

```
import qpformat
# The data are not loaded into memory, only the meta data is read
dataset = qpformat.load_data("/path/to/hologram_image.tif")
# Get the quantitative phase data (a qpimage.QPImage is returned)
qpi = dataset.get_qpimage()
# Get the 2D phase image data as a numpy array
phase = qpi.pha
```

The object `qpi` is an instance of `qpimage.QPImage` which comes with an elaborate set of background correction methods. Note that `qpformat.load_data()` accepts keyword arguments that allow to define the setup metadata as well as the hologram reconstruction parameters.

2.3 Command-line program “qpinfo”

This command-line program allows checking whether a file (or directory) contains quantitative phase data with a file format supported by qpformat.

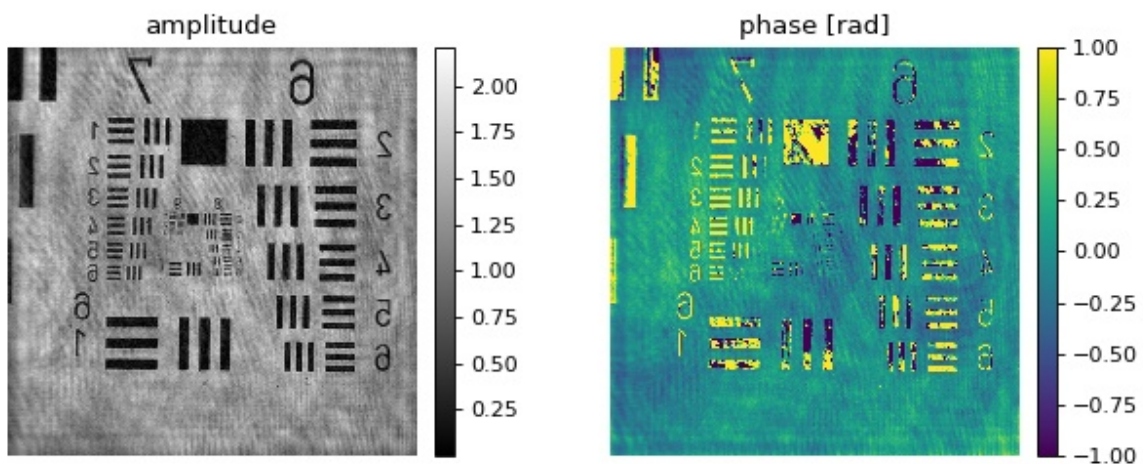
```
usage: qpinfo [-h] path
```

The command yields the type of the format, the corresponding class name in qpformat, as well as the meta data associated with the dataset (e.g. wavelength, pixel size).

3.1 Hologram from tif file

This example illustrates how to retrieve phase and amplitude from a hologram stored as a tif file. The experimental hologram is a U.S. Air Force test target downloaded from the [Submersible Holographic Astrobiology Microscope with Ultraresolution \(SHAMU\) project \[BBL+17\]](#). The values for pixel resolution, wavelength, and reconstruction distance are taken from the corresponding [Python example](#).

The object returned by the `get_qpimage <qpformat.file_formats.dataset.SingleData.get_qpimage()` function is an instance of `qpimage.QPImage` which allows for field refocusing. The refocused QPImage is background-corrected using a polynomial fit to the phase data at locations where the amplitude data is not attenuated (bright regions in the amplitude image).

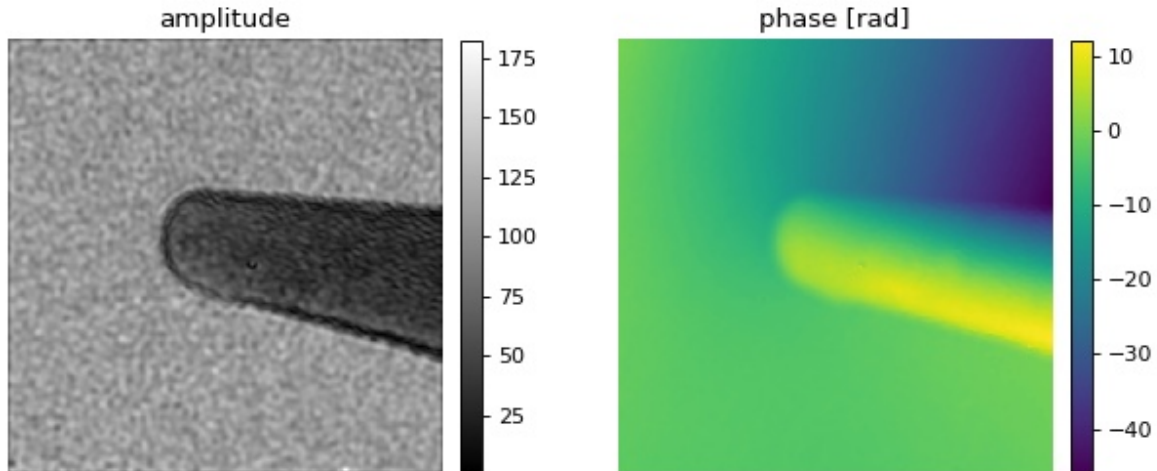


tif_hologram.py

```
1 import urllib.request
2 import os
3
4 import matplotlib.pyplot as plt
5 import qpformat
6
7
8 # load the experimental data
9 dl_loc = "https://github.com/bmorris3/shampoo/raw/master/data/"
10 dl_name = "USAF_test.tif"
11 if not os.path.exists(dl_name):
12     print("Downloading {} ...".format(dl_name))
13     urllib.request.urlretrieve(dl_loc + dl_name, dl_name)
14
15
16 ds = qpformat.load_data(dl_name,
17                         # manually set meta data
18                         meta_data={"pixel size": 3.45e-6,
19                                   "wavelength": 405e-9,
20                                   "medium index": 1},
21                         # set filter size to 1/2 (defaults to 1/3)
22                         # which increases the image resolution
23                         holo_kw={"filter_size": .5})
24
25 # retrieve the qpimage.QPImage instance
26 qpi = ds.get_qpimage()
27 # refocus `qpi` to 0.03685m
28 qpi_foc = qpi.refocus(0.03685)
29 # perform an offset-based amplitude correction
30 qpi_foc.compute_bg(which_data="amplitude",
31                  fit_profile="offset",
32                  fit_offset="mode",
33                  border_px=10,
34                  )
35 # perform a phase correction using
36 # - those pixels that are not dark in the amplitude image (amp_bin) and
37 # - a 2D second order polynomial fit to the phase data
38 amp_bin = qpi_foc.amp > 1 # bright regions
39 qpi_foc.compute_bg(which_data="phase",
40                  fit_profile="poly2o",
41                  from_mask=amp_bin,
42                  )
43
44 # plot results
45 plt.figure(figsize=(8, 3.5))
46 # amplitude
47 ax1 = plt.subplot(121, title="amplitude")
48 map1 = plt.imshow(qpi_foc.amp, cmap="gray")
49 plt.colorbar(map1, ax=ax1, fraction=.0455, pad=0.04)
50 # phase in interval [-1rad, 1rad]
51 ax2 = plt.subplot(122, title="phase [rad]")
52 map2 = plt.imshow(qpi_foc.pha, vmin=-1, vmax=1)
53 plt.colorbar(map2, ax=ax2, fraction=.0455, pad=0.04)
54 # disable axes
55 [ax.axis("off") for ax in [ax1, ax2]]
56 plt.tight_layout()
57 plt.show()
```

3.2 HyperSpy hologram file format

This example demonstrates the import of hologram images in the HyperSpy hdf5 file format. The off-axis electron hologram shows an electrically biased Fe needle [MLFDB15]. The corresponding HyperSpy demo can be found [here](#).



hyperspy_hologram.py

```

1 import urllib.request
2 import os
3
4 import matplotlib.pyplot as plt
5 import qpformat
6
7 # load the experimental data
8 dl_loc = "https://github.com/hyperspy/hyperspy/raw/RELEASE_next_major/" \
9         + "hyperspy/misc/holography/example_signals/"
10 dl_name = "01_holo_Vbp_130V_0V_bin2_crop.hdf5"
11 if not os.path.exists(dl_name):
12     print("Downloading {} ...".format(dl_name))
13     urllib.request.urlretrieve(dl_loc + dl_name, dl_name)
14
15 ds = qpformat.load_data(dl_name,
16                        holo_kw={
17                            # reduces ringing artifacts in the amplitude image
18                            "filter_name": "smooth disk",
19                            # select correct sideband
20                            "sideband": -1,
21                        })
22
23 # retrieve the qpimage.QPImage instance
24 qpi = ds.get_qpimage(0)
25
26 # plot results
27 plt.figure(figsize=(8, 3.5))
28 # amplitude
29 ax1 = plt.subplot(121, title="amplitude")
30 map1 = plt.imshow(qpi.amp, cmap="gray")

```

(continues on next page)

(continued from previous page)

```

31 plt.colorbar(map1, ax=ax1, fraction=.0455, pad=0.04)
32 # phase in interval [-1rad, 1rad]
33 ax2 = plt.subplot(122, title="phase [rad]")
34 map2 = plt.imshow(qpi.pha)
35 plt.colorbar(map2, ax=ax2, fraction=.0455, pad=0.04)
36 # disable axes
37 [ax.axis("off") for ax in [ax1, ax2]]
38 plt.tight_layout()
39 plt.show()

```

3.3 Conversion of external file formats to .npy files

Sometimes the data recorded are not in a file format supported by qpformat or it is not feasible to implement a reader class for a very unique data set. In this example, QPI data, stored as a tuple of files (“*_intensity.txt” and “*_phase.txt”) with commas as decimal separators, are converted to the numpy file format which is supported by qpformat.

This example must be executed with a directory as an command line argument, i.e. `python convert_txt2numpy /path/to/folder/`

`convert_txt2numpy.py`

```

1  import pathlib
2  import sys
3
4  import numpy as np
5
6
7  def get_paths(folder):
8      '''Return *_phase.txt files in `folder`'''
9      folder = pathlib.Path(folder).resolve()
10     files = folder.rglob("*_phase.txt")
11     return sorted(files)
12
13
14  def load_file(path):
15      '''Load a txt data file'''
16      path = pathlib.Path(path)
17      data = path.open().readlines()
18      # remove comments and empty lines
19      data = [l for l in data if len(l.strip()) and not l.startswith("#")]
20      # determine data shape
21      n = len(data)
22      m = len(data[0].strip().split())
23      res = np.zeros((n, m), dtype=np.dtype(float))
24      # write data to array, replacing comma with point decimal separator
25      for ii in range(n):
26          res[ii] = np.array(data[ii].strip().replace(",", ".").split(),
27                             dtype=float)
28
29      return res
30
31  def load_field(path):
32      '''Load QPI data using *_phase.txt files'''
33      path = pathlib.Path(path)

```

(continues on next page)

(continued from previous page)

```

34 phase = load_file(path)
35 inten = load_file(path.parent / (path.name[:-10] + "_intensity.txt"))
36 ampli = np.sqrt(inten)
37 return ampli * np.exp(1j * phase)
38
39
40 if __name__ == "__main__":
41     path = pathlib.Path(sys.argv[-1])
42     if not path.is_dir():
43         raise ValueError("Command line argument must be directory!")
44     # output directory
45     pout = path.parent / (path.name + "_np")
46     pout.mkdir(exist_ok=True)
47     # get input *_phase.txt files
48     files = get_paths(path)
49     # conversion
50     for ff in files:
51         field = load_field(ff)
52         np.save(str(pout / (ff.name[:-10] + ".npy")), field)

```

3.4 Conversion of external holograms to .tif files

Qpformat can load hologram data from .tif image files. If your experimental hologram data are stored in a different file format, you can either request its implementation in qpformat by [creating an issue](#) or you can modify this example script to your needs.

This example must be executed with a directory as an command line argument, i.e. `python convert_txt2tif.py /path/to/folder/`

`convert_txt2tif.py`

```

1  import pathlib
2  import sys
3
4  import numpy as np
5  from skimage.external import tiffimage
6
7  # File names ending with these strings are ignored
8  ignore_endswith = ['.bmp', '.npy', '.opj', '.png', '.pptx', '.py', '.svg',
9                    '.tif', '.txt', '_RIdist', '_parameter', '_parameter_old',
10                   '_phase', 'n_array', 'n_array_real', '~']
11 # uncomment this line to keep background hologram files
12 ignore_endswith += ['_bg']
13
14
15 def get_paths(folder, ignore_endswith=ignore_endswith):
16     '''Return hologram file paths
17
18     Parameters
19     -----
20     folder: str or pathlib.Path
21             Path to search folder
22     ignore_endswith: list
23             List of filename ending strings indicating which

```

(continues on next page)

```
24     files should be ignored.
25     '''
26     folder = pathlib.Path(folder).resolve()
27     files = folder.rglob("*")
28     for ie in ignore_endswith:
29         files = [ff for ff in files if not ff.name.endswith(ie)]
30     return sorted(files)
31
32
33 if __name__ == "__main__":
34     path = pathlib.Path(sys.argv[-1])
35     if not path.is_dir():
36         raise ValueError("Command line argument must be directory!")
37     # output directory
38     pout = path.parent / (path.name + "_tif")
39     pout.mkdir(exist_ok=True)
40     # get input hologram files
41     files = get_paths(path)
42     # conversion
43     for ff in files:
44         # convert image data to uint8 (most image sensors)
45         hol = np.loadtxt(str(ff), dtype=np.uint8)
46         tifout = str(pout / (ff.name + ".tif"))
47         # compress image data
48         tiffimage.imsave(tifout, hol, compress=9)
```


4.1 module-level

```
qpformat.load_data(path, fmt=None, bg_data=None, bg_fmt=None, meta_data={}, holo_kw={},  
                  as_type='float32')
```

Load experimental data

Parameters

- **path** (*str*) – Path to experimental data file or folder
- **fmt** (*str*) – The file format to use (see *file_formats.formats*). If set to *None*, the file format is guessed.
- **bg_data** (*str*) – Path to background data file or *qpimage.QPImage*
- **bg_fmt** (*str*) – The file format to use (see *file_formats.formats*) for the background. If set to *None*, the file format is be guessed.
- **meta_data** (*dict*) – Meta data (see *qpimage.meta.DATA_KEYS*)
- **as_type** (*str*) – Defines the data type that the input data is casted to. The default is “float32” which saves memory. If high numerical accuracy is required (does not apply for a simple 2D phase analysis), set this to double precision (“float64”).

Returns *dataobj* – Object that gives lazy access to the experimental data.

Return type *SeriesData* or *SingleData*

4.2 file format base classes

4.2.1 SeriesData

```
class qpformat.file_formats.SeriesData (path, meta_data={}, holo_kw={},
                                         as_type='float32')
```

Series data file format base class

Parameters

- **path** (*str* or *pathlib.Path*) – Path to the experimental data file.
- **meta_data** (*dict*) – Dictionary containing meta data. see `qpimage.META_KEYS`.
- **as_type** (*str*) – Defines the data type that the input data is casted to. The default is “float32” which saves memory. If high numerical accuracy is required (does not apply for a simple 2D phase analysis), set this to double precision (“float64”).

as_type = None

Enforced dtype via keyword arguments

path = None

`pathlib.Path` to data file or `io.IOBase`

meta_data = None

Enforced metadata via keyword arguments

holo_kw = None

Hologram retrieval; keyword arguments for `qpimage.holo.get_field()`.

background_identifier = None

Unique string that identifies the background data that was set using `set_bg`.

identifier

Return a unique identifier for the given data set

get_identifier (*idx*)

Return an identifier for the data at index *idx*

Changed in version 0.4.2: indexing starts at 1 instead of 0

get_name (*idx*)

Return name of data at index *idx*

Changed in version 0.4.2: indexing starts at 1 instead of 0

get_time (*idx*)

Return time of data at index *idx*

Returns nan if the time is not defined

get_qpimage (*idx*)

Return background-corrected QPImage of data at index *idx*

get_qpimage_raw (*idx*)

Return QPImage without background correction

Note that this method must always return a QPImage instance with the “identifier” metadata key set!

saveh5 (*h5file*, *count=None*, *max_count=None*)

Save the data set as an hdf5 file (qpimage.QPSeries format)

Parameters

- **h5file** (*str*, *pathlib.Path*, or *h5py.Group*) – Where to store the series data
- **max_count** (*count*,) – Can be used to monitor the progress of the algorithm. Initially, the value of *max_count.value* is incremented by the total number of steps. At each step, the value of *count.value* is incremented.

set_bg (*dataset*)

Set background data

Parameters *dataset* (*DataSet*, *qpimage.QPImage*, or *int*) – If the `len(dataset)` matches `len(self)`, then background correction is performed element-wise. Otherwise, `len(dataset)` must be one and is used for all data of *self*.

See also:

[`get_qpimage\(\)`](#) obtain the background corrected QPImage

static verify (*path*)

Verify that *path* has this file format

Returns *True* if the file format matches. The implementation of this method should be fast and memory efficient, because e.g. the “GroupFolder” file format depends on it.

4.2.2 SingleData

```
class qpformat.file_formats.SingleData (path, meta_data={}, holo_kw={},
                                         as_type='float32')
```

Single data file format base class

Parameters

- **path** (*str* or *pathlib.Path*) – Path to the experimental data file.
- **meta_data** (*dict*) – Dictionary containing meta data. see `qpimage.META_KEYS`.
- **as_type** (*str*) – Defines the data type that the input data is casted to. The default is “float32” which saves memory. If high numerical accuracy is required (does not apply for a simple 2D phase analysis), set this to double precision (“float64”).

get_identifier (*idx=0*)

Return an identifier for the data at index *idx*

Changed in version 0.4.2: indexing starts at 1 instead of 0

get_name (*idx=0*)

Return name of data at index *idx*

Changed in version 0.4.2: indexing starts at 1 instead of 0

get_qpimage (*idx=0*)

Return background-corrected QPImage of data at index *idx*

get_qpimage_raw (*idx=0*)

QPImage without background correction

get_time (*idx=0*)

Time of the data

Returns nan if the time is not defined

identifier

Return a unique identifier for the given data set

saveh5 (*h5file*, *count=None*, *max_count=None*)

Save the data set as an hdf5 file (qpimage.QPSeries format)

Parameters

- **h5file** (*str*, *pathlib.Path*, or *h5py.Group*) – Where to store the series data
- **max_count** (*count*,) – Can be used to monitor the progress of the algorithm. Initially, the value of *max_count.value* is incremented by the total number of steps. At each step, the value of *count.value* is incremented.

set_bg (*dataset*)

Set background data

Parameters dataset (*DataSet*, *qpimage.QPImage*, or *int*) – If the `len(dataset)` matches `len(self)`, then background correction is performed element-wise. Otherwise, `len(dataset)` must be one and is used for all data of *self*.

See also:

[*get_qpimage\(\)*](#) obtain the background corrected QPImage

static verify (*path*)

Verify that *path* has this file format

Returns *True* if the file format matches. The implementation of this method should be fast and memory efficient, because e.g. the “GroupFolder” file format depends on it.

4.3 file format readers

All file formats inherit from `qpformat.file_formats.SeriesData` (and/or `qpformat.file_formats.SingleData`).

4.3.1 SeriesFolder

class `qpformat.file_formats.SeriesFolder` (**args*, ***kwargs*)

Folder-based wrapper file format

is_series = `True`

storage_type

The storage type depends on the wrapped file format

files

List of files (only supported file formats)

4.3.2 SeriesHdf5HyperSpy

class `qpformat.file_formats.SeriesHdf5HyperSpy` (*path*, *meta_data={}*, *holo_kw={}*, *as_type='float32'*)

HyperSpy hologram series (HDF5 format)

HyperSpy has its own implementation to read this file format.

```

is_series = True
storage_type = 'hologram'

```

4.3.3 SeriesHdf5Qpimage

```

class qpformat.file_formats.SeriesHdf5Qpimage(*args, **kwargs)
    Qpimage series (HDF5 format)

    is_series = True

    storage_type = 'phase, amplitude'

```

4.3.4 SeriesHdf5QpimageSubjoined

```

class qpformat.file_formats.SeriesHdf5QpimageSubjoined(*args, **kwargs)
    Subjoined qpimage series (HDF5 format), may contain other data

    is_series = True

    storage_type = 'phase, amplitude'

```

4.3.5 SeriesZipTifHolo

```

class qpformat.file_formats.SeriesZipTifHolo(*args, **kwargs)
    Off-axis hologram series (zipped TIFF files)

    The data are stored as multiple TIFF files (qpformat.file_formats.SingleTifHolo) in a zip file.

    is_series = True

    storage_type = 'hologram'

    files
        List of hologram data file names in the input zip file

```

4.3.6 SeriesZipTifPhasics

```

class qpformat.file_formats.SeriesZipTifPhasics(*args, **kwargs)
    Phasics series data (zipped "SID PHA*.tif" files)

    The data are stored as multiple TIFF files (qpformat.file_formats.SingleTifPhasics) in a zip file.

    is_series = True

    storage_type = 'phase, intensity'

    files
        List of Phasics tif file names in the input zip file

```

4.3.7 SingleHdf5Qpimage

```

class qpformat.file_formats.SingleHdf5Qpimage(*args, **kwargs)
    Qpimage single (HDF5 format)

    See the documentation of qpimage for more information.

```

```
is_series = False
storage_type = 'phase,amplitude'
```

4.3.8 SingleNpyNumpy

```
class qpformat.file_formats.SingleNpyNumpy(path, meta_data={}, holo_kw={},
                                           as_type='float32')
```

Numpy complex field or phase data (numpy binary format)

The experimental data given in *path* consist of a single 2D ndarray (no pickled objects). The ndarray is either complex-valued (scattered field) or real-valued (phase).

```
is_series = False
```

```
storage_type
```

Depending on input data type, the storage type is either “field” (complex) or “phase” (real).

4.3.9 SingleTifHolo

```
class qpformat.file_formats.SingleTifHolo(path, meta_data={}, holo_kw={},
                                           as_type='float32')
```

Off-axis hologram image (TIFF format)

```
is_series = False
```

```
storage_type = 'hologram'
```

4.3.10 SingleTifPhasics

```
class qpformat.file_formats.SingleTifPhasics(path, meta_data={}, *args, **kwargs)
Phasics image (“SID PHA*.tif”)
```

Notes

- Only the processed phase data files are supported, i.e. TIFF file names starting with “SID PHA” exported by the commercial Phasics software.
- If the “wavelength” key in *meta_data* is not set (units: [nm]), then the wavelength is extracted from the xml data stored in tag “61238” of the tif file.

```
is_series = False
```

```
storage_type = 'phase,intensity'
```

4.4 exceptions

```
exception qpformat.file_formats.MultipleFormatsNotSupportedError
```

Used when a folder contains series file formats

(see [GitHub issue #1](#))

```
exception qpformat.file_formats.UnknownFileFormatError
```

Used when a file format could not be detected

List of changes in-between qpformat releases.

5.1 version 0.8.0

- feat: allow tracking the progress of `SeriesData.saveh5` using `multiprocessing.Value` objects

5.2 version 0.7.1

- docs: fix missing changelog files

5.3 version 0.7.0

- feat: added command-line entry point “qpinfo”
- feat: support subjoined QPSeries file format
- docs: minor update

5.4 version 0.6.4

- fix: ignore `None` or `nan` values in given meta data

5.5 version 0.6.3

- enh: introduce `BadFileFormatError`

5.6 version 0.6.2

- maintenance release

5.7 version 0.6.1

- tests: fix bad identification of data types

5.8 version 0.6.0

- BREAKING CHANGE: SeriesFolder file format does not load data files recursively anymore

5.9 version 0.5.1

- fix: falsely detected datasets in SeriesFolder file format

5.10 version 0.5.0

- feat: use file modification time as fallback for TIFF files when the file format does not implement *get_time*
- fix: return “nan” instead of “0” when the time is not defined for a measurement

5.11 version 0.4.4

- fix: SeriesFolder file format should not support folders containing no usable data
- fix: verify file format given by user

5.12 version 0.4.3

- fix: “identifier” not always set for generated instances of QPImage

5.13 version 0.4.2

- fix: implement *get_name* method for SeriesFolder format
- enh: start identifier/name indexing at 1 instead of 0

5.14 version 0.4.1

- fix: do not allow intensity values less than zero for SingleTifPhasics

5.15 version 0.4.0

- feat: extract meta data from QPSeries/QPImage data files

5.16 version 0.3.5

- fix: single_tif_phaiscs (SID4Bio) contains two phase images, the second of which is recorded at a different time point than the intensity image. The first image is recorded in wavelengths and not in nanometers and thus is converted using phaiscs metadata first. If this metadata is not available, the second image is used.

5.17 version 0.3.4

- fix: qpimage file formats: override identifiers (clean solution)
- fix: add check for valid meta_data keys
- docs: document attributes of SeriesData

5.18 version 0.3.3

- fix: qpimage file formats: identifiers were not unique, but simply copied from the input hdf5 file

5.19 version 0.3.2

- setup: add qpimage version dependency

5.20 version 0.3.1

- ci: automate PyPI release with travis-ci

5.21 version 0.3.0

- docs: automatically document all file format classes
- docs: add introduction and file format overview
- tests: improve coverage

5.22 version 0.2.1

- fix: regression when loading data from zip file

5.23 version 0.2.0

- drop support for Python 3.5
- fix: SeriesHdf5Qpimage blocked hdf5 file for reading
- fix: background datasets did not get hologram keyword arguments
- feat: allow to cast input data type (qpimage version 0.2.0)

5.24 version 0.1.6

- code cleanup

5.25 version 0.1.5

- fix: raw phasics tif files were not ignored in *SeriesFolder* (#3)
- feat: reduce length of dataset hashes to six chars for user convenience
- feat: switch order of name and index in identifier for user convenience

5.26 version 0.1.4

- feat: new file format for zipped hologram tif files
- feat: add “storage_type” property describing which type of data is stored originally in a dataset
- feat: add hologram file formats: HyperSpy and tif-based
- fix: use hologram keyword arguments to generate dataset identifier

5.27 version 0.1.3

- feat: save memory by hard-linking background image data in QPSeries
- fix: format series and single hdf5: override raw meta data
- fix: include background data in determination of data set identifiers

5.28 version 0.1.2

- feat: change API for SingleData (“idx=0” for user convenience)
- feat: implement SeriesData.saveh5 (export as qpimage.QPSeries) (#2)
- feat: add unique part of file name to SeriesFolder image identifiers (#2)
- feat: extract identifiers from hdf5 files

5.29 version 0.1.1

- feat: support pathlib
- feat: add SeriesData.identifier

5.30 version 0.1.0

- initial release

CHAPTER 6

Bibliography

CHAPTER 7

Indices and tables

- `genindex`
- `modindex`
- `search`

Bibliography

- [BBL+17] Manuel Bedrossian, Casey Barr, Chris A. Lindensmith, Kenneth Neelson, and Jay L. Nadeau. Quantifying microorganisms at low concentrations using digital holographic microscopy (DHM). *Journal of Visualized Experiments*, nov 2017. doi:10.3791/56343.
- [MLFDB15] V. Migunov, A. London, M. Farle, and R. E. Dunin-Borkowski. Model-independent measurement of the charge density distribution along an fe atom probe needle using off-axis electron holography without mean inner potential effects. *Journal of Applied Physics*, 117(13):134301, apr 2015. doi:10.1063/1.4916609.

A

as_type (qpformat.file_formats.SeriesData attribute), 14

B

background_identifier (qpformat.file_formats.SeriesData attribute), 14

F

files (qpformat.file_formats.SeriesFolder attribute), 16

files (qpformat.file_formats.SeriesZipTifHolo attribute), 17

files (qpformat.file_formats.SeriesZipTifPhasics attribute), 17

G

get_identifier() (qpformat.file_formats.SeriesData method), 14

get_identifier() (qpformat.file_formats.SingleData method), 15

get_name() (qpformat.file_formats.SeriesData method), 14

get_name() (qpformat.file_formats.SingleData method), 15

get_qpimage() (qpformat.file_formats.SeriesData method), 14

get_qpimage() (qpformat.file_formats.SingleData method), 15

get_qpimage_raw() (qpformat.file_formats.SeriesData method), 14

get_qpimage_raw() (qpformat.file_formats.SingleData method), 15

get_time() (qpformat.file_formats.SeriesData method), 14

get_time() (qpformat.file_formats.SingleData method), 15

H

holo_kw (qpformat.file_formats.SeriesData attribute), 14

I

identifier (qpformat.file_formats.SeriesData attribute), 14

identifier (qpformat.file_formats.SingleData attribute), 15

is_series (qpformat.file_formats.SeriesFolder attribute), 16

is_series (qpformat.file_formats.SeriesHdf5HyperSpy attribute), 16

is_series (qpformat.file_formats.SeriesHdf5Qpimage attribute), 17

is_series (qpformat.file_formats.SeriesHdf5QpimageSubjoined attribute), 17

is_series (qpformat.file_formats.SeriesZipTifHolo attribute), 17

is_series (qpformat.file_formats.SeriesZipTifPhasics attribute), 17

is_series (qpformat.file_formats.SingleHdf5Qpimage attribute), 17

is_series (qpformat.file_formats.SingleNpyNumpy attribute), 18

is_series (qpformat.file_formats.SingleTifHolo attribute), 18

is_series (qpformat.file_formats.SingleTifPhasics attribute), 18

L

load_data() (in module qpformat), 13

M

meta_data (qpformat.file_formats.SeriesData attribute), 14

MultipleFormatsNotSupportedError, 18

P

path (qpformat.file_formats.SeriesData attribute), 14

S

saveh5() (qpformat.file_formats.SeriesData method), 14

saveh5() (qpformat.file_formats.SingleData method), 16

SeriesData (class in qpformat.file_formats), 14

SeriesFolder (class in qpformat.file_formats), 16
SeriesHdf5HyperSpy (class in qpformat.file_formats), 16
SeriesHdf5Qpimage (class in qpformat.file_formats), 17
SeriesHdf5QpimageSubjoined (class in qpformat.file_formats), 17
SeriesZipTifHolo (class in qpformat.file_formats), 17
SeriesZipTifPhasics (class in qpformat.file_formats), 17
set_bg() (qpformat.file_formats.SeriesData method), 15
set_bg() (qpformat.file_formats.SingleData method), 16
SingleData (class in qpformat.file_formats), 15
SingleHdf5Qpimage (class in qpformat.file_formats), 17
SingleNpyNumpy (class in qpformat.file_formats), 18
SingleTifHolo (class in qpformat.file_formats), 18
SingleTifPhasics (class in qpformat.file_formats), 18
storage_type (qpformat.file_formats.SeriesFolder attribute), 16
storage_type (qpformat.file_formats.SeriesHdf5HyperSpy attribute), 17
storage_type (qpformat.file_formats.SeriesHdf5Qpimage attribute), 17
storage_type (qpformat.file_formats.SeriesHdf5QpimageSubjoined attribute), 17
storage_type (qpformat.file_formats.SeriesZipTifHolo attribute), 17
storage_type (qpformat.file_formats.SeriesZipTifPhasics attribute), 17
storage_type (qpformat.file_formats.SingleHdf5Qpimage attribute), 18
storage_type (qpformat.file_formats.SingleNpyNumpy attribute), 18
storage_type (qpformat.file_formats.SingleTifHolo attribute), 18
storage_type (qpformat.file_formats.SingleTifPhasics attribute), 18

U

UnknownFileFormatError, 18

V

verify() (qpformat.file_formats.SeriesData static method), 15
verify() (qpformat.file_formats.SingleData static method), 16