qpformat Documentation

Release 0.11.0

Paul Müller

Feb 02, 2022

CONTENTS

Qpformat is a Python3 library for opening quantitative phase imaging data file formats. This is the documentation of qpformat version 0.11.0.

CHAPTER

ONE

INTRODUCTION

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
SeriesFolder	*multiple*	Folder-based wrapper file format
SeriesHDF5Sinogram	field	sinograms extracted from Meep/FDTD simulations
SeriesHdf5HyperSpy	hologram	HyperSpy hologram series (HDF5 format)
SeriesHdf5Qpimage	phase,amplitude	Qpimage series (HDF5 format)
SeriesHdf5QpimageSubjoined	phase,amplitude	Subjoined qpimage series (HDF5 format), may contain other data
SeriesZipTifHolo	hologram	Off-axis hologram series (zipped TIFF files)
SeriesZipTifPhasics	phase,intensity	Phasics series data (zipped "SID PHA*.tif" files)
SingleHdf5Qpimage	phase,amplitude	Qpimage single (HDF5 format)
SingleNpyNumpy	*multiple*	Numpy complex field or phase data (numpy binary format)
SingleTifHolo	hologram	Off-axis hologram image (TIFF format)
SingleTifPhasics	phase,intensity	Phasics image ("SID PHA*.tif")

CHAPTER

TWO

GETTING STARTED

2.1 Installing qpformat

qpformat is written in pure Python and supports Python version 3.6 and higher.

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 .`

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).

CHAPTER

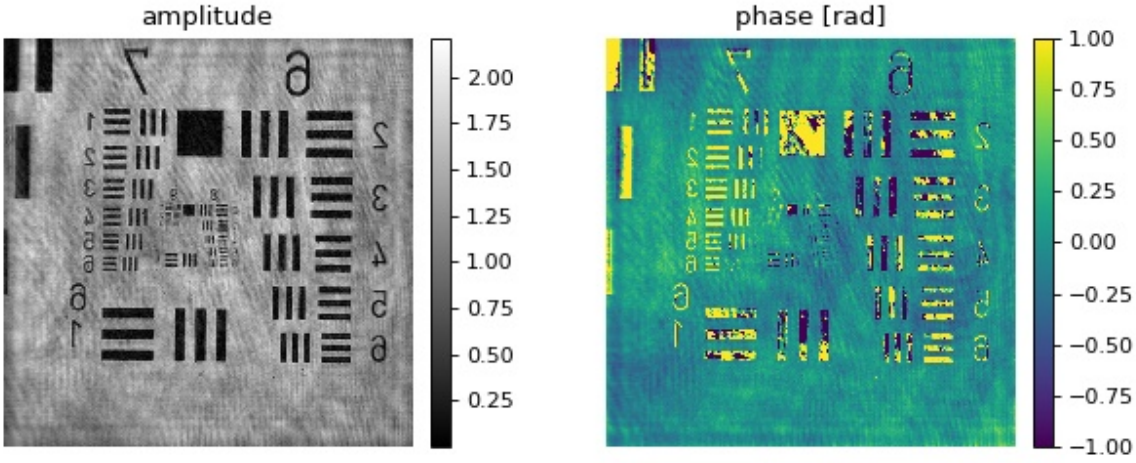
THREE

EXAMPLES

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

```
import urllib.request
import os

import matplotlib.pylab as plt
import qpformat


# load the experimental data
```

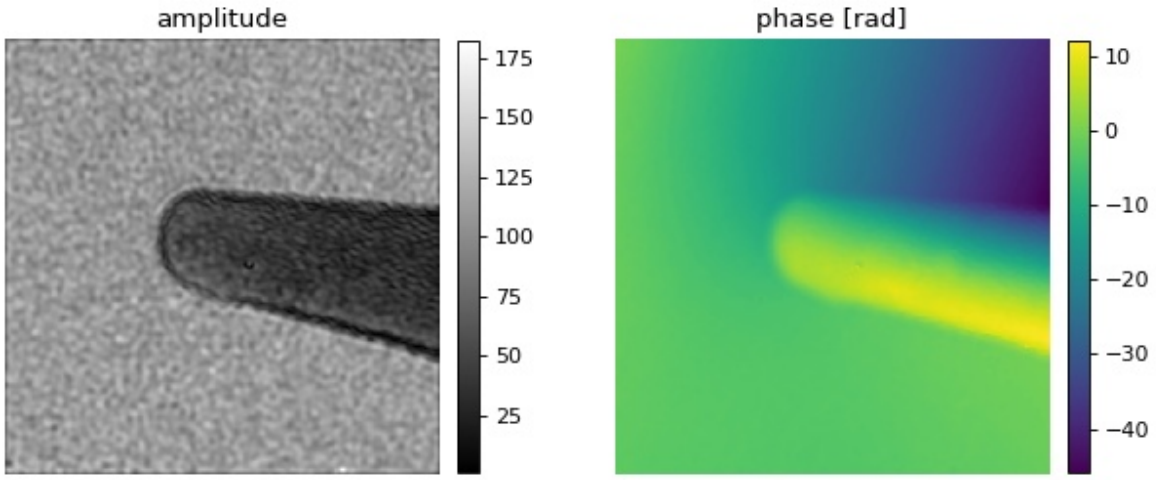
(continues on next page)

(continued from previous page)

```
dl_loc = "https://github.com/bmorris3/shampoo/raw/master/data/"
dl_name = "USAF_test.tif"
if not os.path.exists(dl_name):
    print("Downloading {} ...".format(dl_name))
    urllib.request.urlretrieve(dl_loc + dl_name, dl_name)


ds = qpformat.load_data(dl_name,
                        # manually set meta data
                        meta_data={"pixel size": 3.45e-6,
                                   "wavelength": 405e-9,
                                   "medium index": 1},
                        # set filter size to 1/2 (defaults to 1/3)
                        # which increases the image resolution
                        holo_kw={"filter_size": .5})

# retrieve the qpimage.QPImage instance
qpi = ds.get_qpimage()
# refocus `qpi` to 0.03685m
qpi_foc = qpi.refocus(0.03685)
# perform an offset-based amplitude correction
qpi_foc.compute_bg(which_data="amplitude",
                   fit_profile="offset",
                   fit_offset="mode",
                   border_px=10,
                   )
# perform a phase correction using
# - those pixels that are not dark in the amplitude image (amp_bin) and
# - a 2D second order polynomial fit to the phase data
amp_bin = qpi_foc.amp > 1  # bright regions
qpi_foc.compute_bg(which_data="phase",
                   fit_profile="poly2o",
                   from_mask=amp_bin,
                   )

# plot results
plt.figure(figsize=(8, 3.5))
# ampltitude
ax1 = plt.subplot(121, title="amplitude")
map1 = plt.imshow(qpi_foc.amp, cmap="gray")
plt.colorbar(map1, ax=ax1, fraction=.0455, pad=0.04)
# phase in interval [-1rad, 1rad]
ax2 = plt.subplot(122, title="phase [rad]")
map2 = plt.imshow(qpi_foc.pha, vmin=-1, vmax=1)
plt.colorbar(map2, ax=ax2, fraction=.0455, pad=0.04)
# disable axes
[ax.axis("off") for ax in [ax1, ax2]]
plt.tight_layout()
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

```
import urllib.request
import os

import matplotlib.pylab as plt
import qpformat

# load the experimental data
dl_loc = "https://github.com/hyperspy/hyperspy/raw/RELEASE_next_major/" \
         + "hyperspy/misc/holography/example_signals/"
dl_name = "01_holo_Vbp_130V_0V_bin2_crop.hdf5"
if not os.path.exists(dl_name):
    print("Downloading {} ...".format(dl_name))
    urllib.request.urlretrieve(dl_loc + dl_name, dl_name)

ds = qpformat.load_data(dl_name,
                        holo_kw={
                            # reduces ringing artifacts in the amplitude image
                            "filter_name": "smooth disk",
                            # select correct sideband
                            "sideband": -1,
                            })

# retrieve the qpimage.QPImage instance
qpi = ds.get_qpimage(0)

# plot results
plt.figure(figsize=(8, 3.5))
# ampltitude
```

(continues on next page)

(continued from previous page)

```
ax1 = plt.subplot(121, title="amplitude")
map1 = plt.imshow(qpi.amp, cmap="gray")
plt.colorbar(map1, ax=ax1, fraction=.0455, pad=0.04)
# phase in interval [-1rad, 1rad]
ax2 = plt.subplot(122, title="phase [rad]")
map2 = plt.imshow(qpi.pha)
plt.colorbar(map2, ax=ax2, fraction=.0455, pad=0.04)
# disable axes
[ax.axis("off") for ax in [ax1, ax2]]
plt.tight_layout()
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_txt2npy.py /path/to/folder/`

convert_txt2npy.py

```
import pathlib
import sys

import numpy as np


def get_paths(folder):
    '"Return *_phase.txt files in `folder`"'
    folder = pathlib.Path(folder).resolve()
    files = folder.rglob("*_phase.txt")
    return sorted(files)


def load_file(path):
    '"Load a txt data file"'
    path = pathlib.Path(path)
    data = path.open().readlines()
    # remove comments and empty lines
    data = [ll for ll in data if len(ll.strip()) and not ll.startswith("#")]
    # determine data shape
    n = len(data)
    m = len(data[0].strip().split())
    res = np.zeros((n, m), dtype=np.dtype(float))
    # write data to array, replacing comma with point decimal separator
    for ii in range(n):
        res[ii] = np.array(data[ii].strip().replace(",", ".").split(),
                           dtype=float)
    return res
```

(continues on next page)

(continued from previous page)

```


def load_field(path):
    "Load QPI data using *_phase.txt files"
    path = pathlib.Path(path)
    phase = load_file(path)
    inten = load_file(path.parent / (path.name[:-10] + "_intensity.txt"))
    ampli = np.sqrt(inten)
    return ampli * np.exp(1j * phase)


if __name__ == "__main__":
    path = pathlib.Path(sys.argv[-1])
    if not path.is_dir():
        raise ValueError("Command line argument must be directory!")
    # output directory
    pout = path.parent / (path.name + "_npy")
    pout.mkdir(exist_ok=True)
    # get input *_phase.txt files
    files = get_paths(path)
    # conversion
    for ff in files:
        field = load_field(ff)
        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

```
import pathlib
import sys

import numpy as np
from skimage.external import tifffile

# File names ending with these strings are ignored
# (these are files related to previous analyses)
ignore_endswith = ['.bmp', '.npy', '.opj', '.png', '.pptx', '.py', '.svg',
                   '.tif', '.txt', '_RIdist', '_parameter', '_parameter_2',
                   '_parameter_3', '_parameter_4', '_parameterdrymass',
                   '_parameter_old', '_phase', 'n_array', 'n_array_1',
                   'n_array_drymass1', 'n_array_drymass2', 'n_array_real',
                   '~', '.dat']
# uncomment this line to keep background hologram files
```

(continues on next page)

(continued from previous page)

```
ignore_endswith += ['_bg']


def get_paths(folder, ignore_endswith=ignore_endswith):
    '''Return hologram file paths

    Parameters
    ----------
    folder: str or pathlib.Path
        Path to search folder
    ignore_endswith: list
        List of filename ending strings indicating which
        files should be ignored.
    '''
    folder = pathlib.Path(folder).resolve()
    files = folder.rglob("*")
    for ie in ignore_endswith:
        files = [ff for ff in files if not ff.name.endswith(ie)]
    return sorted(files)


if __name__ == "__main__":
    path = pathlib.Path(sys.argv[-1])
    if not path.is_dir():
        raise ValueError("Command line argument must be directory!")
    # output directory
    pout = path.parent / (path.name + "_tif")
    pout.mkdir(exist_ok=True)
    # get input hologram files
    files = get_paths(path)
    # conversion
    for ff in files:
        # convert image data to uint8 (most image sensors)
        hol = np.loadtxt(str(ff), dtype=np.uint8)
        tifout = str(pout / (ff.name + ".tif"))
        # compress image data
        tifffile.imsave(tifout, hol, compress=9)
```

CHAPTER

FOUR

CODE REFERENCE

4.1 module-level

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

Load experimental data

Parameters

- **path** (*str or pathlib.Path*) – 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 or pathlib.Path*) – 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*)
- **holo_kw** (*dict*) – Keyword arguments for hologram data; See `qpimage.holo.get_field()` for valid keyword arguments.
- **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=None*, *holo_kw=None*, *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").

path
pathlib.Path to data file or io.IOBase

meta_data
Enforced metadata via keyword arguments

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

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

property identifier
Return a unique identifier for the given data set

property shape
Return dataset shape (lenght, image0, image1).

This should be overridden by the subclass, because by default the first qpimage is used for that.

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*

abstract 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*, *qpi_slice=None*, *series_slice=None*, *time_interval=None*, *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
- **qpi_slice** (*tuple of (slice, slice)*) – If not None, only store a slice of each QPImage in *h5file*. A value of None is equivalent to `(slice(0, -1), slice(0, -1))`.
- **series_slice** (*slice*) – If None, save the entire series, otherwise only save the images specified by this slice.
- **time_interval** (*tuple of (float, float)*) – If not None, only stores QPImages that were recorded within the given time interval.

- **count** (*multiprocessing.Value*) – 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.
- **max_count** (*multiprocessing.Value*) – 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.

Notes

The series "identifier" meta data is only set when all of *qpi_slice*, *series_slice*, and *time_interval* are None.

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

abstract 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=None*, *holo_kw=None*, *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*

abstract 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

property identifier
Return a unique identifier for the given data set

saveh5(*h5file*, *qpi_slice=None*, *series_slice=None*, *time_interval=None*, *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
- **qpi_slice** (*tuple of (slice, slice)*) – If not None, only store a slice of each QPImage in *h5file*. A value of None is equivalent to `(slice(0, -1), slice(0, -1))`.
- **series_slice** (*slice*) – If None, save the entire series, otherwise only save the images specified by this slice.
- **time_interval** (*tuple of (float, float)*) – If not None, only stores QPImages that were recorded within the given time interval.
- **count** (*multiprocessing.Value*) – 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.
- **max_count** (*multiprocessing.Value*) – 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.

Notes

The series "identifier" meta data is only set when all of *qpi_slice*, *series_slice*, and *time_interval* are None.

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

property shape
Return dataset shape (lenght, image0, image1).

This should be overridden by the subclass, because by default the first qpimage is used for that.

abstract 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.

path
pathlib.Path to data file or io.IOBase

meta_data
Enforced metadata via keyword arguments

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

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

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

property files
List of files (only supported file formats)

4.3.2 SeriesHDF5SinogramMeep

class qpformat.file_formats.**SeriesHDF5SinogramMeep**(*path*, *meta_data=None*, **args*, ***kwargs*)
sinograms extracted from Meep/FDTD simulations

I introduced this format in 2022 as part of my efforts to make the finite-difference time domain simulations from the ODTbrain manuscript [MSG15] publicly available.

The HDF5 file contains a "background" and a "sinogram" group. The subgroups of "sinogram" are enumerated starting with "0". Each of them contain the complex "field" at a plane behind the scattering phantom as an HDF5 Dataset. The location of the plane (and all other relevant metadata) is stored in the attributes of this Dataset. In the same group, there are also the C++ "simulation_code" and the log "simulation_output" which can be used to reproduce the simulation.

Initialize with default wavelength of 500nm

is_series = True

storage_type = 'field'

4.3.3 SeriesHdf5HyperSpy

class qpformat.file_formats.**SeriesHdf5HyperSpy**(*path*, *meta_data=None*, *holo_kw=None*, *as_type='float32'*)

HyperSpy hologram series (HDF5 format)

HyperSpy has its own implementation to read this file format.

is_series = True

storage_type = 'hologram'

4.3.4 SeriesHdf5Qpimage

class qpformat.file_formats.**SeriesHdf5Qpimage**(**args*, ***kwargs*)

Qpimage series (HDF5 format)

is_series = True

storage_type = 'phase,amplitude'

4.3.5 SeriesHdf5QpimageSubjoined

class qpformat.file_formats.**SeriesHdf5QpimageSubjoined**(**args*, ***kwargs*)

Subjoined qpimage series (HDF5 format), may contain other data

is_series = True

storage_type = 'phase,amplitude'

4.3.6 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'

property files

List of hologram data file names in the input zip file

4.3.7 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'

property files

List of Phasics tif file names in the input zip file

4.3.8 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'

4.3.9 SingleNpyNumpy

class qpformat.file_formats.**SingleNpyNumpy**(*path*, *meta_data=None*, *holo_kw=None*, *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).

4.3.10 SingleTifHolo

class qpformat.file_formats.**SingleTifHolo**(*path*, *meta_data=None*, *holo_kw=None*, *as_type='float32'*)
Off-axis hologram image (TIFF format)

is_series = False

storage_type = 'hologram'

4.3.11 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: [m]), 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

CHAPTER

FIVE

CHANGELOG

List of changes in-between qpformat releases.

5.1 version 0.11.0

- feat: implement FDTD/Meep reader for data from ODTbrain paper
- fix: remove call to deprecated Element.getchildren
- setup: bump qpimage from 0.6.2 to 0.7.4 (new metadata keys)

5.2 version 0.10.9

- ref: minor code cleanup
- setup: remove unneccessary requirements from setup.py

5.3 version 0.10.8

- fix: wavelength could not be extracted for some phasics files (section name “analyse data” vs “analyse data v1”)
- fix: time could not be parsed due to comma instead of dot in phasics meta data

5.4 version 0.10.7

- ci: migrate to GHA
- docs: fix sphinx build
- setup: setup.py test is deprecated

5.5 version 0.10.6

- maintenance release

5.6 version 0.10.5

- setup: bump qpimage from 0.6.1 to 0.6.2
- setup: change dependency of scikit-image to tifffile 2020.5.25
- ref: make code work with latest version of tifffile

5.7 version 0.10.4

- setup: bump qpimage from 0.5.0 to 0.6.1

5.8 version 0.10.3

- fix: missing keyword argument *idx* in *SingleTifHolo.get_time*
- fix: correctly pass precision to phasics and hologram zip file formats
- fix: add file size to dataset identifier hashing process
- example: add conversion script to hdf5 qpimage data

5.9 version 0.10.2

- fix: SeriesFolder did not pass on holo_kw when loading hologram data

5.10 version 0.10.1

- fix: SeriesHdf5Qpimage did not correctly load meta data "time"

5.11 version 0.10.0

- feat: allow to specify a time interval or a series slice in SeriesData.saveh5
- fix: seek to zero before computing a data identifier for an io.IOBase object
- fix: only write series identifier in SeriesData.saveh5 if the keyword arguments for slice extraction are not set
- ref: fix deprecated *.value* (h5py)
- tests: fix date extraction from zip file

5.12 version 0.9.0

- feat: allow to specify a slice in SeriesData.saveh5 (useful when only a specific region needs to be extracted)

5.13 version 0.8.0

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

5.14 version 0.7.1

- docs: fix missing changelog files

5.15 version 0.7.0

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

5.16 version 0.6.4

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

5.17 version 0.6.3

- enh: introduce BadFileFormatError

5.18 version 0.6.2

- maintenance release

5.19 version 0.6.1

- tests: fix bad identification of data types

5.20 version 0.6.0

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

5.21 version 0.5.1

- fix: falsely detected datasets in SeriesFolder file format

5.22 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

5.23 version 0.4.4

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

5.24 version 0.4.3

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

5.25 version 0.4.2

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

5.26 version 0.4.1

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

5.27 version 0.4.0

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

5.28 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 phasics metadata first. If this metadata is not available, the second image is used.

5.29 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

5.30 version 0.3.3

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

5.31 version 0.3.2

- setup: add qpimage version dependency

5.32 version 0.3.1

- ci: automate PyPI release with travis-ci

5.33 version 0.3.0

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

5.34 version 0.2.1

- fix: regression when loading data from zip file

5.35 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)

5.36 version 0.1.6

- code cleanup

5.37 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

5.38 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 identifer

5.39 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

5.40 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

5.41 version 0.1.1

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

5.42 version 0.1.0

- initial release

CHAPTER

SIX

BILBLIOGRAPHY

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[BBL+17] Manuel Bedrossian, Casey Barr, Chris A. Lindensmith, Kenneth Nealson, 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.

[MSG15] Paul Müller, Mirjam Schürmann, and Jochen Guck. ODTbrain: a Python library for full-view, dense diffraction tomography. *BMC Bioinformatics*, 16(1):1–9, 2015. doi:10.1186/s12859-015-0764-0.

INDEX

U

V