DXchange Documentation

Release 0.1.5

Argonne National Laboratory

Nov 02, 2018

Contents

DXchange provides an interface with tomoPy *[B5]* and raw tomographic data collected at different synchrotron facilities including the Data Exchange file format (DXfile) *[A1]*, currently in use at the Advanced Photon Source beamline 2-BM and 32-ID, at the Swiss Light Source Tomcat beamline and at the Elettra SYRMEP beamline *[B3]*.

CHAPTER 1

Features

- Readers for tomographic data files collected at different facilities.
- Writers for different file formats.

CHAPTER 2

Contribute

- Documentation: https://github.com/data-exchange/dxchange/tree/master/doc
- Issue Tracker: https://github.com/data-exchange/dxchange/issues
- Source Code: https://github.com/data-exchange/dxchange

CHAPTER 3

Content

3.1 Install

This section covers the basics of how to download and install DXchange.

Contents:

3.1.1 Installing from source

Clone the DXchange from GitHub repository:

```
git clone https://github.com/data-exchange/dxchange.git dxchange
```

then:

```
cd dxchange
python setup.py install
```

DXchange is dependent on other libraries, listed in the requirements.txt and meta.yaml files.

3.1.2 Installing from Conda/Binstar

First you must have Conda installed, then open a terminal or a command prompt window and run:

```
conda install -c conda-forge dxchange
```

3.1.3 Updating the installation

Data Management is an active project, so we suggest you update your installation frequently. To update the installation run in your terminal:

```
conda update -c conda-forge dxchange
```

For some more information about using Conda, please refer to the docs.

3.2 API reference

dxchange Modules:

3.2.1 dxchange.exchange

Module for describing beamline/experiment specific data recipes.

Functions:

read_als_832(fname[, ind_tomo, normalized, ...])	Read ALS 8.3.2 standard data format.
read_als_832h5(fname[, ind_tomo, ind_flat, ...])	Read ALS 8.3.2 hdf5 file with stacked datasets.
read_anka_topotomo(fname, ind_tomo, ...[, ...])	Read ANKA TOPO-TOMO standard data format.
read_aps_1id(fname[, ind_tomo, proj, sino])	Read APS 1-ID standard data format.
read_aps_2bm(fname[, proj, sino])	Read APS 2-BM standard data format.
read_aps_5bm(fname[, sino])	Read APS 5-BM standard data format.
read_aps_7bm(fname[, proj, sino])	Read APS 7-BM standard data format.
read_aps_13bm(fname, format[, proj, sino])	Read APS 13-BM standard data format.
read_aps_13id(fname[, group, proj, sino])	Read APS 13-ID standard data format.
read_aps_26id(image_directory, ind_tomo, ...)	Read APS 26-ID tomography data from a stack of xrm files.
read_aps_32id(fname[, exchange_rank, proj, ...])	Read APS 32-ID standard data format.
read_aus_microct(fname, ind_tomo, ind_flat, ...)	Read Australian Synchrotron micro-CT standard data format.
read_diamond_l12(fname, ind_tomo[, proj])	Read Diamond Light Source L12 (JEEP) standard data format.
read_elettra_syrmep(fname, ind_tomo, ...[, ...])	Read Elettra SYRMEP standard data format.
read_esrf_id19(fname[, proj, sino])	Read ESRF ID-19 standard data format.
read_lnls_imx(folder[, proj, sino])	Read LNLS IMX standard data format.
read_petraIII_p05(fname, ind_tomo, ind_flat, ...)	Read Petra-III P05 standard data format.
read_sls_tomcat(fname[, ind_tomo, proj, sino])	Read SLS TOMCAT standard data format.

dxchange.exchange.**read_als_832**(*fname*, *ind_tomo=None*, *normalized=False*, *proj=None*, *sino=None*)

Read ALS 8.3.2 standard data format.

Parameters

- **fname** (*str*) – Path to file name without indices and extension.
- **ind_tomo** (*list of int, optional*) – Indices of the projection files to read.
- **normalized** (*boolean, optional*) – If False, darks and flats will not be read. This should only be used for cases where tomo is already normalized. 8.3.2 has a plugin that normalization is preferred to be done with prior to tomopy reconstruction.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_als_832h5**(*fname*, *ind_tomo=None*, *ind_flat=None*, *ind_dark=None*, *proj=None*, *sino=None*)

Read ALS 8.3.2 hdf5 file with stacked datasets.

Parameters

- **fname** (*str*) – Path to hdf5 file.
- **ind_tomo** (*list of int, optional*) – Indices of the projection files to read.
- **ind_flat** (*list of int, optional*) – Indices of the flat field files to read.
- **ind_dark** (*list of int, optional*) – Indices of the dark field files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.
- *list of int* – Indices of flat field data within tomography projection list

dxchange.exchange.**read_anka_topotomo**(*fname*, *ind_tomo*, *ind_flat*, *ind_dark*, *proj=None*, *sino=None*)

Read ANKA TOPO-TOMO standard data format.

Parameters

- **fname** (*str*) – Path to data folder name without indices and extension.
- **ind_tomo** (*list of int*) – Indices of the projection files to read.
- **ind_flat** (*list of int*) – Indices of the flat field files to read.
- **ind_dark** (*list of int, optional*) – Indices of the dark field files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_aps_1id**(*fname*, *ind_tomo=None*, *proj=None*, *sino=None*)
Read APS 1-ID standard data format.

Parameters

- **fname** (*str*) – Path to file name without indices and extension.
- **ind_tomo** (*list of int, optional*) – Indices of the projection files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_aps_2bm**(*fname*, *proj=None*, *sino=None*)
Read APS 2-BM standard data format.

Parameters

- **fname** (*str*) – Path to hdf5 file.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_aps_5bm**(*fname*, *sino=None*)
Read APS 5-BM standard data format.

Parameters

- **fname** (*str*) – Path to data folder.
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_aps_7bm**(*fname*, *proj=None*, *sino=None*)
Read APS 7-BM standard data format.

Parameters

- **fname** (*str*) – Path to hdf5 file.

- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *array* – Projection angles in radian.

dxchange.exchange.**read_aps_8bm**(*image_directory*, *ind_tomo*, *ind_flat*, *image_file_pattern=u'image_00000.xrm'*, *flat_file_pattern=u'ref_00000.xrm'*, *proj=None*, *sino=None*)

Read APS 8-BM tomography data from a stack of xrm files.

Parameters

- **image_directory** (*str*) – Path to data folder name without indices and extension.
- **ind_tomo** (*list of int*) – Indices of the projection files to read.
- **ind_flat** (*list of int*) – Indices of the flat field files to read.
- **image_file_pattern** (*string*) – Specify how the projection files are named.
- **flat_file_pattern** (*string*) – Specify how the flat reference files are named.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *dictionary* – Image set metadata.

dxchange.exchange.**read_aps_13bm**(*fname*, *format*, *proj=None*, *sino=None*)

Read APS 13-BM standard data format. Searches directory for all necessary files, and then combines the separate flat fields.

Parameters

- **fname** (*str*) – Path to hdf5 file.
- **format** (*str*) – Data format. 'spe' or 'netcdf4'
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns *ndarray* – 3D tomographic data.

dxchange.exchange.**read_aps_13id**(*fname*, *group=u'/xrfmap/roimap/sum_cor'*, *proj=None*, *sino=None*)

Read APS 13-ID standard data format.

Parameters

- **fname** (*str*) – Path to hdf5 file.
- **group** (*str, optional*) – Path to the group inside hdf5 file where data is located.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns *ndarray* – 3D tomographic data.

dxchange.exchange.**read_aps_26id**(*image_directory*, *ind_tomo*, *ind_flat*, *image_file_pattern=u'image_00000.xrm'*, *flat_file_pattern=u'ref_00000.xrm'*, *proj=None*, *sino=None*)

Read APS 26-ID tomography data from a stack of xrm files.

Parameters

- **fname** (*str*) – Path to data folder name without indices and extension.
- **ind_tomo** (*list of int*) – Indices of the projection files to read.
- **ind_flat** (*list of int*) – Indices of the flat field files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *dictionary* – Image set metadata.

dxchange.exchange.**read_aps_32id**(*fname*, *exchange_rank=0*, *proj=None*, *sino=None*, *dtype=None*)

Read APS 32-ID standard data format.

Parameters

- **fname** (*str*) – Path to hdf5 file.
- **exchange_rank** (*int, optional*) – exchange_rank is added to "exchange" to point tomopy to the data to reconstruct. if rank is not set then the data are raw from the detector and are located under exchange = "exchange/...", to process data that are the result of some intemedite processing step then exchange_rank = 1, 2, ... will direct tomopy to process "exchange1/...",
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)
- **dtype** (*numpy datatype, optional*) – Convert data to this datatype on read if specified.

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.
- *ndarray* – 1D theta in radian.

dxchange.exchange.**read_aus_microct**(*fname*, *ind_tomo*, *ind_flat*, *ind_dark*, *proj=None*, *sino=None*)

Read Australian Synchrotron micro-CT standard data format.

Parameters

- **fname** (*str*) – Path to data folder.
- **ind_tomo** (*list of int*) – Indices of the projection files to read.
- **ind_flat** (*list of int*) – Indices of the flat field files to read.
- **ind_dark** (*list of int*) – Indices of the dark field files to read.

- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_diamond_l12**(*fname*, *ind_tomo*, *proj=None*)

Read Diamond Light Source L12 (JEEP) standard data format.

Parameters

- **fname** (*str*) – Path to data folder.
- **ind_tomo** (*list of int*) – Indices of the projection files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.

dxchange.exchange.**read_elettra_syrmep**(*fname*, *ind_tomo*, *ind_flat*, *ind_dark*, *proj=None*, *sino=None*)

Read Elettra SYRMEP standard data format.

Parameters

- **fname** (*str*) – Path to data folder.
- **ind_tomo** (*list of int*) – Indices of the projection files to read.
- **ind_flat** (*list of int*) – Indices of the flat field files to read.
- **ind_dark** (*list of int*) – Indices of the dark field files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_esrf_id19**(*fname*, *proj=None*, *sino=None*)

Read ESRF ID-19 standard data format.

Parameters

- **fname** (*str*) – Path to edf file.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.

- *ndarray* – 3D dark field data.

dxchange.exchange.**read_lnls_imx**(*folder*, *proj=None*, *sino=None*)

Read LNLS IMX standard data format.

Parameters

- **folder** (*str*) – Path to sample folder (containing tomo.h5, flat.h5, dark.h5)
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_petraIII_p05**(*fname*, *ind_tomo*, *ind_flat*, *ind_dark*, *proj=None*, *sino=None*)

Read Petra-III P05 standard data format.

Parameters

- **fname** (*str*) – Path to data folder.
- **ind_tomo** (*list of int*) – Indices of the projection files to read.
- **ind_flat** (*list of int*) – Indices of the flat field files to read.
- **ind_dark** (*list of int*) – Indices of the dark field files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

dxchange.exchange.**read_sls_tomcat**(*fname*, *ind_tomo=None*, *proj=None*, *sino=None*)

Read SLS TOMCAT standard data format.

Parameters

- **fname** (*str*) – Path to file name without indices and extension.
- **ind_tomo** (*list of int, optional*) – Indices of the projection files to read.
- **proj** (*{sequence, int}, optional*) – Specify projections to read. (start, end, step)
- **sino** (*{sequence, int}, optional*) – Specify sinograms to read. (start, end, step)

Returns

- *ndarray* – 3D tomographic data.
- *ndarray* – 3D flat field data.
- *ndarray* – 3D dark field data.

3.2.2 dxchange.reader

Module for importing data files.

Functions:

read_edf(fname[, slc])	Read data from edf file.
read_hdf5(fname, dataset[, slc, dtype, shared])	Read data from hdf5 file from a specific group.
read_netcdf4(fname, group[, slc])	Read data from netcdf4 file from a specific group.
read_npy(fname[, slc])	Read binary data from a `.npy` file.
read_spe(fname[, slc])	Read data from spe file.
read_fits(fname[, fixdtype])	Read data from fits file.
read_tiff(fname[, slc])	Read data from tiff file.
read_tiff_stack(fname, ind[, digit, slc])	Read data from stack of tiff files in a folder.
read_hdf5_stack(h5group, dname, ind[, ...])	Read data from stacked datasets in a hdf5 file
read_xrm(fname[, slice_range])	Read data from xrm file.
read_xrm_stack(fname, ind[, slc])	Read data from stack of xrm files in a folder.
read_txrm(file_name[, slice_range])	Read data from a .txrm file, a compilation of .xrm files.

dxchange.reader.**read_edf**(*fname*, *slc=None*)

Read data from edf file.

Parameters

- **fname** (*str*) – String defining the path of file or file name.
- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), ... , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Data.

dxchange.reader.**read_hdf5**(*fname*, *dataset*, *slc=None*, *dtype=None*, *shared=False*)

Read data from hdf5 file from a specific group.

Parameters

- **fname** (*str*) – String defining the path of file or file name.
- **dataset** (*str*) – Path to the dataset inside hdf5 file where data is located.
- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), ... , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.
- **dtype** (*numpy datatype (optional)*) – Convert data to this datatype on read if specified.
- **shared** (*bool (optional)*) – If True, read data into shared memory location. Defaults to True.

Returns *ndarray* – Data.

dxchange.reader.**read_netcdf4**(*fname*, *group*, *slc=None*)

Read data from netcdf4 file from a specific group.

Parameters

- **fname** (*str*) – String defining the path of file or file name.
- **group** (*str*) – Variable name where data is stored.

- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), … , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Data.

dxchange.reader.**read_npy**(*fname*, *slc=None*)
Read binary data from a `.npy` file.

Parameters

- **fname** (*str*) – String defining the path of file or file name.
- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), … , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Data.

dxchange.reader.**read_spe**(*fname*, *slc=None*)
Read data from spe file.

Parameters

- **fname** (*str*) – String defining the path of file or file name.
- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), … , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Data.

dxchange.reader.**read_fits**(*fname*, *fixdtype=True*)
Read data from fits file.

Parameters **fname** (*str*) – String defining the path of file or file name.

Returns *ndarray* – Data.

dxchange.reader.**read_tiff**(*fname*, *slc=None*)
Read data from tiff file.

Parameters

- **fname** (*str*) – String defining the path of file or file name.
- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), … , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Output 2D image.

dxchange.reader.**read_tiff_stack**(*fname*, *ind*, *digit=None*, *slc=None*)
Read data from stack of tiff files in a folder.

Parameters

- **fname** (*str*) – One of the file names in the tiff stack.
- **ind** (*list of int*) – Indices of the files to read.
- **digit** (*int*) – (Deprecated) Number of digits used in indexing stacked files.
- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), … , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Output 3D image.

dxchange.reader.**read_xrm**(*fname*, *slice_range=None*)
Read data from xrm file.

Parameters

- **fname** (*str*) – String defining the path of file or file name.
- **slice_range** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), ... , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Output 2D image.

dxchange.reader.**read_xrm_stack**(*fname*, *ind*, *slc=None*)
Read data from stack of xrm files in a folder.

Parameters

- **fname** (*str*) – One of the file names in the tiff stack.
- **ind** (*list of int*) – Indices of the files to read.
- **slc** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), ... , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns *ndarray* – Output 3D image.

dxchange.reader.**read_txrm**(*file_name*, *slice_range=None*)
Read data from a .txrm file, a compilation of .xrm files.

Parameters

- **file_name** (*str*) – String defining the path of file or file name.
- **slice_range** (*sequence of tuples, optional*) – Range of values for slicing data in each axis. ((start_1, end_1, step_1), ... , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix.

Returns

- *ndarray* – Array of 2D images.
- *dictionary* – Dictionary of metadata.

dxchange.reader.**read_hdf5_stack**(*h5group*, *dname*, *ind*, *digit=4*, *slc=None*, *out_ind=None*)
Read data from stacked datasets in a hdf5 file

Parameters

- **fname** (*str*) – One of the dataset names in the dataset stack
- **ind** (*list of int*) – Indices of the datasets to be read
- **digit** (*int*) – (Deprecated) Number of digits indexing the stacked datasets
- **slc** (*{sequence, int}*) – Range of values for slicing data. ((start_1, end_1, step_1), ... , (start_N, end_N, step_N)) defines slicing parameters for each axis of the data matrix
- **out_ind** (*list of int, optional*) – Outer level indices for files with two levels of indexing. i.e. [name_000_000.tif, name_000_001.tif, ..., name_000_lmn.tif, name_001_lmn.tif, ..., ..., name_fgh_lmn.tif]

dxchange.reader.**read_file_list**(*file_list*)
Read data from stack of image files in a folder.

Parameters **file_list** (*list of str*) – List of file names to read, in order

3.2.3 dxchange.writer

Module for data exporting data files.

Functions:

write_dxf(data[, fname, axes, dtype, overwrite])	Write data to a data exchange hdf5 file.
write_hdf5(data[, fname, gname, dname, ...])	Write data to hdf5 file in a specific group.
write_npy(data[, fname, dtype, overwrite])	Write data to a binary file in NumPy .npy format.
write_tiff(data[, fname, dtype, overwrite])	Write image data to a tiff file.
write_tiff_stack(data[, fname, dtype, axis, ...])	Write data to stack of tiff file.

dxchange.writer.**write_dxf**(*data*, *fname=u'tmp/data.h5'*, *axes=u'theta:y:x'*, *dtype=None*, *overwrite=False*)

Write data to a data exchange hdf5 file.

Parameters

- **data** (*ndarray*) – Array data to be saved.
- **fname** (*str*) – File name to which the data is saved. .h5 extension will be appended if it does not already have one.
- **axes** (*str*) – Attribute labels for the data array axes.
- **dtype** (*data-type, optional*) – By default, the data-type is inferred from the input data.
- **overwrite** (*bool, optional*) – if True, overwrites the existing file if the file exists.

dxchange.writer.**write_hdf5**(*data*, *fname=u'tmp/data.h5'*, *gname=u'exchange'*, *dname=u'data'*, *dtype=None*, *overwrite=False*, *appendaxis=None*, *maxsize=None*)

Write data to hdf5 file in a specific group.

Parameters

- **data** (*ndarray*) – Array data to be saved.
- **fname** (*str*) – File name to which the data is saved. .h5 extension will be appended if it does not already have one.
- **gname** (*str, optional*) – Path to the group inside hdf5 file where data will be written.
- **dname** (*str, optional*) – Name for dataset where data will be written.
- **dtype** (*data-type, optional*) – By default, the data-type is inferred from the input data.
- **overwrite** (*bool, optional*) – if True, overwrites the existing file if the file exists.
- **appendaxis** (*int, optional*) – Axis where data is to be appended to. Must be given when creating a resizable dataset.
- **maxsizee** (*int, optional*) – Maximum size that the dataset can be resized to along the given axis.

dxchange.writer.**write_npy**(*data*, *fname=u'tmp/data.npy'*, *dtype=None*, *overwrite=False*)

Write data to a binary file in NumPy .npy format.

Parameters

- **data** (*ndarray*) – Array data to be saved.
- **fname** (*str*) – File name to which the data is saved. `.npy` extension will be appended if it does not already have one.

`dxchange.writer.`**`write_tiff`**(*data*, *fname=u'tmp/data.tiff'*, *dtype=None*, *overwrite=False*)

Write image data to a tiff file.

Parameters

- **data** (*ndarray*) – Array data to be saved.
- **fname** (*str*) – File name to which the data is saved. `.tiff` extension will be appended if it does not already have one.
- **dtype** (*data-type, optional*) – By default, the data-type is inferred from the input data.
- **overwrite** (*bool, optional*) – if True, overwrites the existing file if the file exists.

`dxchange.writer.`**`write_tiff_stack`**(*data*, *fname=u'tmp/data.tiff'*, *dtype=None*, *axis=0*, *digit=5*, *start=0*, *overwrite=False*)

Write data to stack of tiff file.

Parameters

- **data** (*ndarray*) – Array data to be saved.
- **fname** (*str*) – Base file name to which the data is saved. `.tiff` extension will be appended if it does not already have one.
- **dtype** (*data-type, optional*) – By default, the data-type is inferred from the input data.
- **axis** (*int, optional*) – Axis along which stacking is performed.
- **start** (*int, optional*) – First index of file in stack for saving.
- **digit** (*int, optional*) – Number of digits in indexing stacked files.
- **overwrite** (*bool, optional*) – if True, overwrites the existing file if the file exists.

3.3 Examples

Below are examples for reading tomographic data sets from different facilities and process them with TomoPy *[B5]*.

3.3.1 Anka TopoTomo

This section contains a script to read the Anka TopoTomo tomography dataset and reconstruct it with tomoPy.

Download file: `rec_anka.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the Anka topo-tomo tomography data as
original tiff.
"""

from __future__ import print_function
import tomopy
```

(continues on next page)

(continued from previous page)

```
import dxchange

if __name__ == '__main__':
    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/'

    proj_start = 0
    proj_end = 1800
    flat_start = 0
    flat_end = 100
    dark_start = 0
    dark_end = 100

    ind_tomo = range(proj_start, proj_end)
    ind_flat = range(flat_start, flat_end)
    ind_dark = range(dark_start, dark_end)

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the Anka tiff raw data.
    proj, flat, dark = dxchange.read_anka_topotomo(fname, ind_tomo, ind_flat,
                                                   ind_dark, sino=(start, end))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024,
                                    ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.2 Australian Synchrotron

This section contains a script to read the Australian Synchrotron Facility tomography dataset and reconstruct it with tomoPy.

Download file: `rec_australian.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the Australian Synchrotron Facility
data as original tiff.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':

    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/'

    proj_start = 0
    proj_end = 1801
    flat_start = 0
    flat_end = 10
    dark_start = 0
    dark_end = 10

    ind_tomo = range(proj_start, proj_end)
    ind_flat = range(flat_start, flat_end)
    ind_dark = range(dark_start, dark_end)

    # Select the sinogram range to reconstruct.
    start = 290
    end = 294

    # Read the Australian Synchrotron Facility data
    proj, flat, dark = dxchange.read_aus_microct(fname, ind_tomo, ind_flat, ind_dark,␣
↪sino=(start, end))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024, ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/aus_')
```

3.3.3 ALS 8.3.2

This section contains a script to read the als 8.3.2 tomography dataset and reconstruct it with tomoPy.

Download file: `rec_als.py` and `rec_als_hdf5.py`

3.3.4 Elettra Syrmep

This section contains a script to read the Elettra syrmep tomography dataset and reconstruct it with tomoPy.

Download file: `rec_elettra.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the Elettra syrmep data as original tiff.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':

    # Set path to the CT data to reconstruct.
    fname = 'data_dir/'

    proj_start = 1
    proj_end = 1801
    flat_start = 1
    flat_end = 11
    dark_start = 1
    dark_end = 11

    ind_tomo = range(proj_start, proj_end)
    ind_flat = range(flat_start, flat_end)
    ind_dark = range(dark_start, dark_end)

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the Elettra syrmep
    proj, flat, dark = dxchange.read_elettra_syrmep(fname, ind_tomo, ind_flat, ind_
↪dark, sino=(start, end))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0], 0, 180)

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024, ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

```

(continues on next page)

(continued from previous page)

```
    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.5 ESRF ID-19

This section contains a script to read the ESRF ID-19 tomography dataset and reconstruct it with tomoPy.

Download file: `rec_esrf.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the ESRF tomography data as original edf
files.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':
    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/'

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the ESRF ID-19 raw data.
    proj, flat, dark = dxchange.read_esrf_id19(fname, sino=(start, end))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024,
                                    ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')
```

(continues on next page)

(continued from previous page)

```
    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.6 APS 1-ID

This section contains a script to read the APS 1-ID tomography dataset and reconstruct it with tomoPy.

Download file: `rec_aps_1id.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the APS 1-ID tomography data as original tiff.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':

    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/sample_name_prefix'

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the APS 1-ID raw data.
    proj, flat, dark = dxchange.read_aps_1id(fname, sino=(start, end))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024, ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.7 APS 5-BM

This section contains a script to read the APS 5-BM tomography dataset and reconstruct it with tomoPy.

Download file: `rec_aps_5bm.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the APS 5-BM data as original xmt.
xmt are 16 bit unsigned integer tiff file that requires a byte swap before
being processed.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':

    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/'

    # Select the sinogram range to reconstruct.
    start = 290
    end = 294

    # Read the APS 5-BM raw data
    proj, flat, dark = dxchange.read_aps_5bm(fname, sino=(start, end))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # remove stripes
    proj = tomopy.remove_stripe_fw(proj,level=7,wname='sym16',sigma=1,pad=True)

    # Set rotation center.
    rot_center = proj.shape[2] / 2.0
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.8 APS 8-BM

This section contains a script to read the X-radia XRM tomography dataset and reconstruct it with tomoPy.

Download file: `rec_aps_8bm.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the xrm tomography data from
the original stack of xrm. To use rename the xrm data as
radios/image_00000.xrm and flats/ref_00000.xrm
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':
    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/'

    proj_start = 0
    proj_end = 1800
    flat_start = 0
    flat_end = 100

    ind_tomo = range(proj_start, proj_end)
    ind_flat = range(flat_start, flat_end)

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the APS 8-BM raw data.
    proj, flat, metadata = dxchange.read_aps_8bm(fname, ind_tomo, ind_flat,
                                                 sino=(start, end))

    # make the darks
    dark = np.zeros((1, proj.shape[1], proj.shape[2]))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024,
                                    ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

```

(continues on next page)

(continued from previous page)

```
    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.9 APS 13-BM

This section contains a script to read the APS 13-BM tomography dataset and reconstruct it with tomoPy.

Download file: `rec_aps_13bm.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the APS 13-BM tomography
data as original netcdf files. To use, change fname to just
the file name (e.g. 'sample[2].nc' would be 'sample'.
Reconstructed dataset will be saved as float32 netcdf3.
"""
import glob
import numpy as np
import tomopy as tp
import dxchange as dx

from netCDF4 import Dataset

if __name__ == '__main__':
    ## Set path (without file suffix) to the micro-CT data to reconstruct.
    fname = 'data_dir/sample'

    ## Import Data.
    proj, flat, dark, theta = dx.exchange.read_aps_13bm(fname, format = 'netcdf4')

    ## Flat-field correction of raw data.
    proj = tp.normalize(proj, flat = flat, dark = dark)

    ## Additional flat-field correction of raw data to negate need to mask.
    proj = tp.normalize_bg(proj, air = 10)

    ## Set rotation center.
    rot_center = tp.find_center_vo(proj)
    print('Center of rotation: ', rot_center)

    tp.minus_log(proj, out = proj)

    # Reconstruct object using Gridrec algorith.
    rec = tp.recon(proj, theta, center = rot_center, sinogram_order = False,␣
↪algorithm = 'gridrec', filter_name = 'hann')
    rec = tp.remove_nan(rec)

    ## Writing data in netCDF3 .volume.
    ncfile = Dataset('filename.volume', 'w', format = 'NETCDF3_64BIT', clobber = True)
    NX = ncfile.createDimension('NX', rec.shape[2])
```

(continues on next page)

(continued from previous page)

```
    NY = ncfile.createDimension('NY', rec.shape[1])
    NZ = ncfile.createDimension('NZ', rec.shape[0])
    volume = ncfile.createVariable('VOLUME', 'f4', ('NZ','NY','NX'))
    volume[:] = rec
    ncfile.close()
```

3.3.10 APS 26-ID

This section contains a script to read the X-radia XRM tomography dataset and reconstruct it with tomoPy.

Download file: `rec_aps_26id.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the xrm tomography data from
the original stack of xrm. To use rename the xrm data as
radios/image_00000.xrm and flats/ref_00000.xrm
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':
    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/'

    proj_start = 0
    proj_end = 1800
    flat_start = 0
    flat_end = 100

    ind_tomo = range(proj_start, proj_end)
    ind_flat = range(flat_start, flat_end)

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the APS 26-ID raw data.
    proj, flat, metadata = dxchange.read_aps_26id(fname, ind_tomo, ind_flat,
                                                  sino=(start, end))

    # make the darks
    dark = np.zeros((1, proj.shape[1], proj.shape[2]))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
```

(continues on next page)

(continued from previous page)

```
    rot_center = tomopy.find_center(proj, theta, init=1024,
                                    ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.11 APS 2-BM & 32-ID

This section contains a script to read the APS 2-BM and 32-ID tomography dataset and reconstruct it with tomoPy.

Download file: `rec_aps_32id_full.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct TXM data set.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':

    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/sample.h5'

    # Select sinogram range to reconstruct.
    start = 0
    end = 16

    # Read APS 32-ID raw data.
    proj, flat, dark, theta = dxchange.read_aps_32id(fname, sino=(start, end))

    # If data collection angles is not defined in the hdf file then set it as equally␣
↪spaced between 0-180 degrees.
    if (theta is None):
        theta = tomopy.angles(proj.shape[0])
    else:
        pass

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
```

(continues on next page)

(continued from previous page)

```
    rot_center = tomopy.find_center(proj, theta, ind=0, init=1024, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.12 Petra III P05

This section contains a script to read the Petra III P05 tomography dataset and reconstruct it with tomoPy.

Download file: `rec_petraIII.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the PetraIII P05 tomography data as original␣
↪tiff.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':

    # Set path to the micro-CT data to reconstruct.
    fname = '/data_dir/sample_name00_0000/'

    proj_start = 0
    proj_end = 1441
    flat_start = 0
    flat_end = 20
    dark_start = 0
    dark_end = 20

    ind_tomo = range(proj_start, proj_end)
    ind_flat = range(flat_start, flat_end)
    ind_dark = range(dark_start, dark_end)

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the Petra III P05
    proj, flat, dark = dxchange.read_petraIII_p05(fname, ind_tomo, ind_flat, ind_dark,
↪ sino=(start, end))
```

(continues on next page)

(continued from previous page)

```
    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024, ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/petra_')
```

3.3.13 SLS Tomcat

This section contains a script to read the Swiss Light Source tomcat tomography dataset and reconstruct it with tomoPy.

Download file: `rec_tomcat.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the Swiss Light Source TOMCAT tomography
data as original tiff.
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':
    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/sample_name_prefix'

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # Read the APS 1-ID raw data.
    proj, flat, dark = dxchange.read_sls_tomcat(fname, sino=(start, end))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0], 0, 180)

    # Flat-field correction of raw data.
```

(continues on next page)

(continued from previous page)

```
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024,
                                    ind=0, tol=0.5)
    print("Center of rotation:", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

3.3.14 X-radia XRM

This section contains a script to read the X-radia XRM tomography dataset and reconstruct it with tomoPy.

Download file: `rec_xradia_xrm.py`

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
TomoPy example script to reconstruct the xrm tomography data from
the original stack of xrm. To use rename the xrm data as
radios/image_00000.xrm and flats/ref_00000.xrm
"""

from __future__ import print_function
import tomopy
import dxchange

if __name__ == '__main__':
    # Set path to the micro-CT data to reconstruct.
    fname = 'data_dir/'

    proj_start = 0
    proj_end = 1800
    flat_start = 0
    flat_end = 100

    ind_tomo = range(proj_start, proj_end)
    ind_flat = range(flat_start, flat_end)

    # Select the sinogram range to reconstruct.
    start = 0
    end = 16

    # APS 26-ID has an x-radia system collecting raw data as xrm.
    proj, flat, metadata = dxchange.read_aps_26id(fname, ind_tomo, ind_flat,
```

(continues on next page)

(continued from previous page)

```
                                                sino=(start, end))

    # make the darks
    dark = np.zeros((1, proj.shape[1], proj.shape[2]))

    # Set data collection angles as equally spaced between 0-180 degrees.
    theta = tomopy.angles(proj.shape[0])

    # Flat-field correction of raw data.
    proj = tomopy.normalize(proj, flat, dark)

    # Find rotation center.
    rot_center = tomopy.find_center(proj, theta, init=1024,
                                    ind=0, tol=0.5)
    print("Center of rotation: ", rot_center)

    proj = tomopy.minus_log(proj)

    # Reconstruct object using Gridrec algorithm.
    rec = tomopy.recon(proj, theta, center=rot_center, algorithm='gridrec')

    # Mask each reconstructed slice with a circle.
    rec = tomopy.circ_mask(rec, axis=0, ratio=0.95)

    # Write data as stack of TIFs.
    dxchange.write_tiff_stack(rec, fname='recon_dir/recon')
```

For a repository of experimental and simulated data sets please check TomoBank *[B4]*.

3.4 Credits

3.4.1 Citations

We kindly request that you cite the following article *[A1]* if you use DXchange.

3.4.2 References

Bibliography

[A1] De Carlo F, Gursoy D, Marone F, Rivers M, Parkinson YD, Khan F, Schwarz N, Vine DJ, Vogt S, Gleber SC, Narayanan S, Newville M, Lanzirotti T, Sun Y, Hong YP, and Jacobsen C. Scientific data exchange: a schema for hdf5-based storage of raw and analyzed data. *Journal of Synchrotron Radiation*, 21(6):1224–1230, 2014.

[B1] The EPICS control system. http://www.aps.anl.gov/epics/. Accessed: 2016-03-12.

[B2] The UDUNITS at UNIDATA. http://www.unidata.ucar.edu/software/udunits/. Accessed: 2016-03-12.

[B3] Francesco Brun, Serena Pacilè, Agostino Accardo, Georgios Kourousias, Diego Dreossi, Lucia Mancini, Giuliana Tromba, and Roberto Pugliese. Enhanced and flexible software tools for x-ray computed tomography at the italian synchrotron radiation facility elettra. *Fundam. Inform.*, 141(2-3):233–243, 2015. URL: http://dx.doi.org/10.3233/FI-2015-1273, doi:10.3233/FI-2015-1273.

[B4] Francesco De Carlo, Doga Gursoy, Daniel Jackson Ching, Kees Joost Batenburg, Wolfgang Ludwig, Lucia Mancini, Federica Marone, Rajmund Mokso, Daniel M. Pelt, Jan Sijbers, and Mark Rivers. Tomobank: a tomographic data repository for computational x-ray science. *Measurement Science and Technology*, 2017. URL: https://doi.org/10.1088/1361-6501/aa9c19.

[B5] Gürsoy D, De Carlo F, Xiao X, and Jacobsen C. Tomopy: a framework for the analysis of synchrotron tomographic data. *Journal of Synchrotron Radiation*, 21(5):1188–1193, 2014.

[B6] De Carlo F, Gursoy D, Marone F, Rivers M, Parkinson YD, Khan F, Schwarz N, Vine DJ, Vogt S, Gleber SC, Narayanan S, Newville M, Lanzirotti T, Sun Y, Hong YP, and Jacobsen C. Scientific data exchange: a schema for hdf5-based storage of raw and analyzed data. *Journal of Synchrotron Radiation*, 21(6):1224–1230, 2014.

[B7] The HDF Group. The HDF Dump. http://www.hdfgroup.org/HDF5/doc/RM/Tools.html#Tools-Dump. Accessed: 2016-03-12.

[B8] The HDF Group. The HDF File Format. http://www.hdfgroup.org/HDF5. Accessed: 2016-03-12.

[B9] The HDF Group. The HDF viewer. http://www.hdfgroup.org/hdf-java-html/hdfview. Accessed: 2016-03-12.

[B10] Filipe R. N. C. Maia. The CXI File Format. https://github.com/FilipeMaia/CXI/raw/master/cxi_file_format.pdf. Accessed: 2016-03-12.

[B11] Filipe R. N. C. Maia. The Coherent X-ray Imaging Data Bank. http://cxidb.org/cxi.html. Accessed: 2016-03-12.

[B12] Ulrik Pedersen, Arthur Glowacki, Alan Greer, and Mark Rivers. Area Detector HDF plugin. http://cars.uchicago.edu/software/epics/NDFileHDF5.html. Accessed: 2016-03-12.

[B13] Mark Rivers. Area Detector. http://cars9.uchicago.edu/software/epics/areaDetector.html. Accessed: 2016-03-12.

Python Module Index

d

Index