nexgen Documentation

Release 0.9.3

Diamond Light Source - Scientific Software

May 16, 2024

CONTENTS:

- Free software: BSD license
- Documentation: https://nexgen.readthedocs.io.

This package provides a set of tools to generate NeXus files following the NXmx Gold Standard.

CHAPTER

ONE

USAGE

Nexgen is a Python package that provides a set of tools to write NeXus files for experiments at Diamond Light Source, following the NXmx application definition for macromolecular crystallography. This is meant to include all relevant experiment metadata required to process the datasets, including detector and goniometer description.

1.1 Installation

Nexgen can be installed using pip.

```
pip install nexgen
```

Note: This project is under development.

1.2 Command line tools

This package started out as an easy way to quickly generate NeXus files from scratch along with blank HDF5 datasets using command line tools.

Parsing The freephil package is used for parsing metadata from the command line.

1.2.1 Getting help

Every command line tool in the nexgen package has a help message that explains how to use it and what the options are. This help message will be printed by using the option *-h*, or *–help*, and each subcommand also has an help message detailing its specific options.

```
copy_nexus --help
```

```
generate_nexus demo -h
```

1.2.2 Show PHIL parameters

In addition to the help message, it is possible to take a look at the list of phil parameters that can/need to be passed to the command line generator.

```
generate_nexus 3 -c
```

It is also possible to view more details about the Phil parameters and definition attributes by setting the *attributes_level* parameter with the *-a* argument. Th default value is set to 0, which will only show names and default values of the parameters.

```
generate_nexus 1 -c -a 2
```

1.2.3 Creating a new .phil file

Writing the full list of parameters on the command line each time can be time consuming, not to mention subject to typing errors and the like. For this purpose, it is possible to generate one reusable Phil file containing the beamline description and those values from the experiment metadata that can be considered constant.

Nexgen already includes Phil files for some MX beamlines at Diamond Light Source, which can be viewed and downloaded by running `nexgen_phil` with the `list` and `get` options. For example, the command

```
nexgen_phil list
```

will return a list of the .phil files currently available, and che chosen file can be downloaded by running:

```
nexgen_phil get paramfile.phil -o  /path/to/directory
```

In case a .phil file for a specific beamline is not in the list, it is possible to either download a blank template (also listed) to fill in manually or create on using the `new` option. While this is a bit more cumbersome, it has the advantage of only needing to write most of the parameters once. Once the file is created it can be parsed by `generate_nexus`, eg.

```
generate_nexus 2 -i paramfile.phil output.master_filename=File.nxs input.vds_
↪writer=dataset
```

To access the help message for `nexgen_phil`:

```
nexgen_phil -h
```

1.2.4 Generating new NeXus files

- For an existing dataset

```
generate_nexus 1 beamline.phil input.datafile=File_00*.h5 input.snaked=True␣
↪\
goniometer.starts=0,0,0,0 goniometer.ends=0,0,1,2 goniometer.increments=0,0,
↪0.1,0.2  \
detector.exposure_time=0.095 detector.beam_center=989.8,1419 detector.
↪overload=65535 \
detector.starts=0,140 detector.ends=0,140 beam.wavelength=0.4859
```

- From scratch, along with blank data (demo)

```
generate_nexus 2 -i/-e beamline.phil output.master_filename=File.nxs input.
↪vds_writer=dataset (etc...)
```

- For an existing dataset which also has a meta.h5 file

```
generate_nexus 3 beamline.phil input.metafile=File_meta.h5 input.vds_
↪writer=dataset output.master_filename=/path/to/File.nxs
```

Note: This functionality will only work properly for Eiger and Tristan detectors.

1.2.5 Generating NXmx-like NeXus files for Electron Diffraction

Example usage for a dataset collected on Dectris Singla 1M detector using a phil parser:

```
ED_nexus singla-phil ED_Singla.phil input.datafiles=FILE_data_*.h5 goniometer.starts=0,0,
↪0,0 \
goniometer.ends=900,0,0,0 goniometer.increments=1,0,0,0 detector.starts=400 detector.
↪beam_center=1,1 \
-m FILE_master.h5
```

The instrument name and source are defined by the values parsed from source, which are shown in the following dictionary:

```
source = {
    "name": "Diamond Light Source",
    "short_name": "DLS",
    "type": "Electron Source",
    "beamline_name": "eBic",
    "probe": "electron",
}
```

Note: As of version *0.6.28*, the source type to go in the NXSource base class has been updated to *Electron Source*.

To specify a more specific name for the */entry/instrument/name* field, the following command can be added to the command line:

```
source.facility_id="DIAMOND MICROSCOPE"
```

which will result in the instrument name being set to *DIAMOND MICROSCOPE eBic* instead of *DIAMOND eBic*.

The downside of this option is that the external links to the data will now be saved using absolute paths instead of relative.

Example usage for a dataset collected on Dectris Singla 1M detector without the phil parser (new as of version *0.7.3*):

```
ED_nexus singla FILE_master.h5 400 -e 0.099 -wl 0.02 -bc 1 1 --axis-name alpha --axis-
↪start 0.0 --axis-inc 0.11
```

For both CLI tools, in case there is a need to save the NeXus file in a different location than the data files:

```
-o /path/to/new/directory
```

1.2.6 Copying NeXus files

- Copy a nexus file in full, or just parts of it. T

  This tool will create a new file File_copy.nxs, in order to avoid modifying the orifinal data, with just the requested metadata.

  ```
  copy_nexus gen input.original_nexus=File.nxs input.simple_copy=True
  ```

  ```
  copy_nexus gen original_nexus=File.nxs data_filename=File_0001.h5␣
  ↪skip=NXdata skip=NXsample
  ```

- Copy metadata from a Tristan NeXus file to NXmx format.

  The main application fo this tool is to copy the necessary metadata to a new NeXus file following the NXmx format after binning event data into images. The default *experiment_type* for copying Tristan metadata is set to rotation; when dealing with a single image, this value can be set to stationary like in the example below.

  ```
  copy_nexus tristan tristan_nexus=Tristan_img.nxs data_filename=Tristan_img_
  ↪0001.h5 experiment_type=stationary
  ```

CHAPTER

TWO

BEAMLINE SPECIFIC UTILITIES

Nexgen is currently being used for some specific applications at beamlines I19-2 and I24 at DLS.

2.1 Time resolved collections on I19-2

Where GDA is not in use, a NXmx format NeXus files writer is available for time-resolved Eiger/Tristan collections.

2.1.1 Example usage

Example I: Rotation scan with Tristan

```
"""
This example calls the nexus writer for a collection using Tristan detector.

Note that in this case the axes start and end positions need to be passed to the writer
and this can be done by defining the following namedtuples:
   axes = namedtuple("axes", ("id", "start", "end"))
    det_axes = namedtuple("det_axes", ("id", "start"))
"""

from nexgen.beamlines.I19_2_nxs import nexus_writer

from datetime import datetime
from collections import namedtuple
from pathlib import Path

axes = namedtuple("axes", ("id", "start", "end"))
det_axes = namedtuple("det_axes", ("id", "start"))

axes_list = [
    axes("omega", 0, 10),
    axes("kappa", 0, 0),
    axes("phi", -90, -90),
    axes("sam_z", 0, 0),
    axes("sam_y", 1, 1),
    axes("sam_x", 0, 0),
]

det_ax_list = [
```

(continues on next page)

(continued from previous page)

```
    det_axes("two_theta", 90),
    det_axes("det_z", 100),
]

nexus_writer(
    meta_file=Path("/path/to/file_meta.h5"),
    detector_name="tristan",
    scan_axis="omega",
    start_time=datetime.now(),
    exposure_time=60.0,
    transmission=1.0,
    wavelength=0.6,
    beam_center=[1000., 1200.],
    gonio_pos=axes_list,
    det_pos=det_ax_list,
)
```

Example II: Rotation scan with Eiger

```
"""
This example calls the nexus writer for a collection using Eiger detector.

Note that in this case there's no need to pass the axes positions as those can be read␣
↪from
the config written to the _meta.h5 file at the arming of the detector.
"""

from nexgen.beamlines.I19_2_nxs import nexus_writer

from datetime import datetime
from pathlib import Path

nexus_writer(
    meta_file=Path("/path/to/file_meta.h5"),
    detector_name="eiger",
    scan_axis="phi",
    start_time=datetime.now(),
    exposure_time=60.0,
    transmission=1.0,
    wavelength=0.6,
    beam_center=[1000., 1200.],
)
```

2.2 Serial crystallography

- I19-2: Fixed target SSX with Tristan detector.
- **I24: serial crystallography with Eiger detector**
  - Still shots (or extruder)
  - Fixed target
  - 3D grid scan

2.2.1 Example usage

Example 1: grid scan on I24

```
"This example calls the SSX writer for a fixed_target experiment on I24."

from nexgen.beamlines.I24_Eiger_nxs import ssx_eiger_writer
from datetime import datetime

beam_x = 1590.7
beam_y = 1643.7

D = 1.480   # Detector distance passed in mm
t = 0.01    # Exposure time passed in s

# Example of chip_dict (form beamline I24) with minimum required values needed for␣
↪goniometer computations.
chip_dict = {
    'X_NUM_STEPS':    [11, 20],
    'Y_NUM_STEPS':    [12, 20],
    'X_STEP_SIZE':    [13, 0.125],
    'Y_STEP_SIZE':    [14, 0.125],
    'X_START':        [16, 0],
    'Y_START':        [17, 0],
    'Z_START':        [18, 0],
    'X_NUM_BLOCKS':   [20, 8],
    'Y_NUM_BLOCKS':   [21, 8],
    'X_BLOCK_SIZE':   [24, 3.175],
    'Y_BLOCK_SIZE':   [25, 3.175],
    'N_EXPOSURES':    [30, 1],
    'PUMP_REPEAT':    [32, 0],
}

ssx_eiger_writer(
    "/path/to/dataset",     # visitpath
    "Expt1_00",    # filename root
    "I24",      # beamline
    "fixed_target",     # experiment type
    pump_status=True,
    num_imgs=1600,
    beam_center=[beam_x, beam_y],
    det_dist=D,
```

(continues on next page)

(continued from previous page)

```
    start_time=datetime.strptime("2022-09-09T14:19:27", "%Y-%m-%dT%H:%M:%S"),
    stop_time=datetime.now(),
    exp_time=t,
    transmission=1.,
    wavelength=0.67019,
    flux=None,
    pump_exp=None,
    pump_delay=0.001,
    chip_info=chip_dict,
    chipmap="/path/to/chip.map/file",
)
```

Example 2: grid scan on I19-2 using Tristan10M

```
"This example calls the SSX writer for a simple time-resolved pump-probe experiment on a␣
↪full chip using Tristan."

from nexgen.beamlines.SSX_Tristan_nxs import ssx_tristan_writer
from datetime import datetime

beam_x = 1590.7
beam_y = 1643.7

D = 0.5     # Detector distance passed in mm
t = 0.002   # Exposure time passed in s

write_nxs(
    "/path/to/dataset",
    "Expt1_00",
    "I19-2",
    exp_time=t,
    det_dist=D,
    beam_center=[beam_x, beam_y],
    transmission=1.,
    wavelength=0.649,
    start_time=datetime.now(),
    stop_time=None,
    chip_info=chip_dict,
    chipmap=None,
)
```

2.3 I19-2 CLI

2.3.1 Example usage

Write a NeXus file for a Tristan collection using a GDA-generated xml file containing the beamline information:

```
I19_nexus 1 Expt_00_meta.h5 Expt.xml tristan 300 0.649 1590.7 1643.7 --start 2022-09-
↪09T10:26:32Z --stop 2022-09-09T10:31:32Z
```

Manually generate a NeXus file for a dataset collected on Eiger detector using the metadata recorded inside the meta file:

```
I19_nexus 2 Expt1_00_meta.h5 eiger 0.02 -tr 100 --use-meta
```

If the *–use-meta* flag is not passed, the writer will not look up the axes/beam_center/wavelength information in the meta file. This will then need to be passed from the commang line:

```
I19_nexus gen Expt1_00_meta.h5 eiger 0.095 -wl 0.485 -bc 989.8 1419 --det-axes det_z --
↪det-start 140 --axes omega phi --ax-start -90 -130.5 --ax-inc 0 0.1 -tr 5 -n 75
```

Note: Only the goniometer/detector axes that have values and increments different from 0 need to be passed to the command line. If –scan-axis is not passed, it will default to 'phi'. If -bc (beam_center) is not passed, in the absence of a meta file it will default to (0, 0)

2.4 SSX CLI

2.4.1 Example usage

Write a NeXus file for a serial collection on Eiger detector on beamline I24 at DLS:

```
SSX_nexus eiger dummy_00_meta.h5 I24 fixed-target 1600 -det 500 -tr 1.0 -wl 0.649 -bc␣
↪1590.7 1643.7 -e 0.002 -p --chipmap testchip.map
```

CHAPTER

THREE

API

3.1 Defining the various parts of a nexus file

class nexgen.utils.**Point3D**(*x*, *y*, *z*)

Coordinates in 3D space.

3.1.1 Axes

Utilities for axes definition

class nexgen.nxs_utils.axes.**Axis**(*name: str*, *depends: str*, *transformation_type:* TransformationType, *vector:* Point3D | *Tuple[float, float, float]*, *start_pos: float = 0.0*, *increment: float = 0.0*, *num_steps: int = 0*, *offset:* Point3D | *Tuple[float, float, float] = (0.0, 0.0, 0.0)*)

Bases: object

Define an axis object for goniometer or detector.

name

Axis name.

Type
str

depends

Name of the axis it depends on.

Type
str

transformation_type

Rotation or translation.

Type
TransformationType

vector

Axis vector.

Type
Point3D | Tuple

start_pos

Start position of axis. Defaults to 0.0.

Type
float, optional

increment

Scan step size if the axis moves. Defaults to 0.0.

Type
float, optional

num_steps

Number of scan points. Defaults to 0.0.

Type
int, optional

offset

Axis offset. Defaults to (0.0, 0.0, 0.0).

Type
Point3D | Tuple, optional

Properties:
units (str): Defined depending on transformation type: deg or mm. end_pos (float): Last point recorded in a scan colletion. Calculated from start_pos, increment and num_steps, 1-indexed. is_scan (bool): Whether axis is a scan axis.

class nexgen.nxs_utils.axes.**TransformationType**(*value*)

Bases: str, Enum

Define axis transformation type - ROTATION - TRANSLATION

3.1.2 Scans

Utilities to look for scan axes and calculate scan ranges from a list of Axis objects.

nexgen.nxs_utils.scan_utils.**calculate_scan_points**(*axis1: Axis*, *axis2: Axis | None = None*, *snaked: bool = True*, *rotation: bool = False*, *tot_num_imgs: int | None = None*) → Dict[str, _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]]

Calculate the scan range for a linear/grid scan or a rotation scan from the number of images (steps) to be written in each direction.

When dealing with a rotation axis, if there are multiple images but no rotation scan, return the axis start position repeated as many times as the number of images - either defined by the num_steps attribute of the Axis object or passed as tot_num_imgs.

Parameters

- **axis1** (Axis) – Axis object describing the axis involved in a scan.

- **axis2** (*Axis, optional*) – Axis object describing the second axis involved in a scan. Only necessary for a grid scan. Defaults to None.
- **snaked** (*bool, optional*) – If True, scanspec will "draw" a grid where the second axis is snaked. It will be ignored for a rotation scan. Defaults to True.
- **rotation** (*bool, optional*) – Tell the function to calculate a rotation scan. Defaults to False.
- **tot_num_imgs** (*int, optional*) – Total number of images. Only used for oscillation axis when there is no rotation. It will be ignored otherwise. Defaults to None.

Raises

- ***ScanAxisError*** – If the passed axis has the wrong transformation type.
- **ValueError** – For a rotation axis with no rotation, if the number of images is missing.

Returns
A dictionary of ("axis_name": axis_range) key-value pairs.

Return type
Dict[str, ArrayLike]

nexgen.nxs_utils.scan_utils.**identify_grid_scan_axes**(*axes_list: List[Axis]*) → List[str]

Identify the scan axes for a translation linear/grid scan.

Parameters
axes_list (*List[Axis]*) – List of axes objects associated to goniometer axes.

Raises
ScanAxisNotFoundError – If no axes have been passed.

Returns
List of strings identifying the linear/grid scan axes. If no axes are identified, it will return an empty list.

Return type
scan_axis (List[str])

nexgen.nxs_utils.scan_utils.**identify_osc_axis**(*axes_list: List[Axis]*, *default: str = 'omega'*) → str

Identify the rotation scan_axis.

This function identifies the scan axis from the list passed as argument. The scan axis is the one where start and end value are not the same. If there is only one rotation axis, that is the one returned. In the case scan axis cannot be identified, a default value is arbitrarily assigned.

Parameters

- **axes_list** (*List[Axis]*) – List of axes objects associated to goniometer axes.
- **default** (*str, optional*) – String to deafult to in case scan axis is not found. Defaults to "omega".

Raises

- ***ScanAxisNotFoundError*** – If no axes have been passed.
- **ValueError** – If more than one rotation axis seems to move.

Returns
String identifying the rotation scan axis.

Return type
scan_axis (str)

class nexgen.nxs_utils.scan_utils.**GridScanOptions**(*axes_order*, *snaked*)

Options for defining a grid scan

axes_order

List of axes in order of (fast, slow).

snaked

Boolean to say whether it's a snaked scan.

exception nexgen.nxs_utils.scan_utils.**ScanAxisNotFoundError**(*errmsg*)

exception nexgen.nxs_utils.scan_utils.**ScanAxisError**(*errmsg*)

3.1.3 Goniometer

Object definition for goniometer.

class nexgen.nxs_utils.goniometer.**Goniometer**(*axes: List[Axis]*, *scan: Dict[str, _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]] | None = None*)

Goniometer definition.

axes_list

List of axes making up the goniometer, including their vectors and positions.

scan

The scan executed during the collection, could be a rotation or a grid scan. If not passed can be updated from the axes.

define_scan_axes_for_event_mode(*end_position: float | None = None*) → Tuple[Dict, Dict]

Define oscillation and/or grid scan ranges for event-mode collections.

define_scan_from_goniometer_axes(*grid_scan_options: GridScanOptions | None = None*, *scan_direction: ScanDirection = ScanDirection.POSITIVE*, *update: bool = True*) → Tuple[Dict, Dict]

Define oscillation and/or grid scan ranges for image data collections.

get_number_of_scan_points()

Get the number of scan points from the defined scan.

3.1.4 Detector

Object definition for detectors.

class nexgen.nxs_utils.detector.**CetaDetector**(*description: str, image_size: ~typing.List[float] | ~typing.Tuple[float], pixel_size: ~typing.List[str | float] = <factory>, sensor_material: str = 'Si', sensor_thickness: str = '0.014mm', detector_type: str = 'CMOS', overload: int = 1000000, underload: int = -1000*)

Bases: DataClassJsonMixin

Define a Ceta-D detector.

class nexgen.nxs_utils.detector.**Detector**(*detector_params: EigerDetector | TristanDetector | SinglaDetector | JungfrauDetector | CetaDetector, detector_axes: List[Axis], beam_center: List[float], exposure_time: float, module_vectors: List[Point3D] | List[Tuple[float, float, float]]*)

Bases: `object`

Detector definition.

detector_params

The detector parameters, unique to each detector type.

detector_axes

The axes where the detector lays, their start positions and vectors in mcstas coordinates.

beam_center

The beam center position, in pixels.

exp_time

The collection exposure time, in seconds.

module

The detector module definition, with fast_axis and slow_axis directions, in mcstas.

get_detector_description() → str

Get detector description string.

get_detector_mode() → str

Data type collected by the detector. If no mode specified in detector parameters, defaults to images.

get_module_info()

Write the module information to a dictionary.

class nexgen.nxs_utils.detector.**DetectorModule**(*fast_axis: Tuple[float, float, float] | Point3D, slow_axis: Tuple[float, float, float] | Point3D, module_offset: str = '1'*)

Bases: `DataClassJsonMixin`

A class to define the axes of a detector module.

fast_axis

Vector defining the fast_axis direction.

Type
Tuple | *Point3D*

slow_axis

Vector defining the slow_axis direction.

Type
Tuple | *Point3D*

class nexgen.nxs_utils.detector.**EigerDetector**(*description: str, image_size: ~typing.List[int] | ~typing.Tuple[int, int], sensor_material: ~typing.Literal['Si', 'CdTe'], overload: int, underload: int, pixel_size: ~typing.List[str | float] = <factory>, detector_type: str = 'Pixel'*)

Bases: `DataClassJsonMixin`

Define a Dectris Eiger detector.

`description`

Detector description.

Type
str

`image_size`

Dimensions in pixels, passed in the order (slow, fast) axis.

Type
List | Tuple

`sensor_material`

Either Si or CdTe, on the material depends the sensor_thickness.

Type
str

`overload`

Saturation value for the detector, data is invalid above this value.

Type
int

`underload`

Lowest value measurable for the detector, data is invalid below this value.

Type
int

`pixel_size`

Size of each detector pixel in both directions, order should be (x, y). Defaults to a pixel size of ['0.075mm', '0.075mm']

Type
List[str], optional

`detector_type`

Description of type of detector. Defaults to 'Pixel'.

Type
str, optional

Properties:
sensor_thickness (str): Defined depending on the sensor material: 0.450mm for Si, 0.750mm CdTe. constants (Dict): Dictionary of meta file locations to create the external links to fields such as pixel_mask, flatfield and bit_depth_readout.

`class nexgen.nxs_utils.detector.JungfrauDetector(`*description: str, image_size: ~typing.List[int] | ~typing.Tuple[int, int], sensor_material: str = 'Si', sensor_thickness: str = '0.320mm', overload: int = 1000000, underload: int = -10, pixel_size: ~typing.List[str | float] = <factory>, detector_type: str = 'Pixel'*`)`

Bases: `DataClassJsonMixin`

Define a Dectris Jungfrau detector.

description

Detector description.

Type
str

image_size

Dimensions in pixels, passed in the order (slow, fast) axis.

Type
List | Tuple

sensor_material

Sensor material. Defaults to Si.

Type
str

sensor_thickness

Sensor thickness. Defaults to 0.320mm

Type
str

overload

Saturation value for the detector, data is invalid above this value. Defaults to 1000000.

Type
int

underload

Lowest value measurable for the detector, data is invalid below this value. Defaults to -10.

Type
int

pixel_size

Size of each detector pixel in both directions, order should be (x, y). Defaults to a pixel size of [‘0.075mm’, ‘0.075mm’]

Type
List[str], optional

detector_type

Description of type of detector. Defaults to ‘Pixel’.

Type
str, optional

Properties:
constants (Dict): Dictionary of meta file locations to create the external links to fields such as pixel_mask, flatfield and bit_depth_readout.

class nexgen.nxs_utils.detector.**SinglaDetector**(*description: str, image_size: ~typing.List[int] | ~typing.Tuple[int, int], sensor_material: str = 'Si', sensor_thickness: str = '0.450mm', overload: int = 199996, underload: int = -1, pixel_size: ~typing.List[str | float] = <factory>, detector_type: str = 'HPC'*)

Bases: DataClassJsonMixin

Define a Dectris Singla detector.

description

Detector description.

Type
str

image_size

Dimensions in pixels, passed in the order (slow, fast) axis.

Type
List | Tuple

sensor_material

Sensor material. Defaults to Si.

Type
str

sensor_thickness

Sensor thickness. Defaults to 0.450mm

Type
str

overload

Saturation value for the detector, data is invalid above this value. Defaults to 199996.

Type
int

underload

Lowest value measurable for the detector, data is invalid below this value. Defaults to -1.

Type
int

pixel_size

Size of each detector pixel in both directions, order should be (x, y). Defaults to a pixel size of [‘0.075mm’, ‘0.075mm’]

Type
List[str], optional

detector_type

Description of type of detector. Defaults to ‘HPC’.

Type
str, optional

Properties:
constants (Dict): Dictionary of meta file locations to create the external links to fields such as pixel_mask, flatfield and bit_depth_readout.

class nexgen.nxs_utils.detector.**TristanDetector**(*description: str, image_size: ~typing.List[int] | ~typing.Tuple[int, int], sensor_material: str = 'Si', sensor_thickness: str = '0.5mm', pixel_size: ~typing.List[str | float] = <factory>, detector_type: str = 'Pixel', mode: ~typing.Literal['events', 'images'] = 'events'*)

Bases: DataClassJsonMixin

Define a Tristan detector.

description

Detector description.

Type
str

image_size

Dimensions in pixels, passed in the order (slow, fast) axis.

Type
List | Tuple

sensor_material

Sensor material. Defaults to Si.

Type
str

sensor_thickness

Sensor thickness. Defaults to 0.5mm

Type
str

pixel_size

Size of each detector pixel in both directions, order should be (x, y). Defaults to a pixel size of ['5.5e-05m', '5.5e-05m']

Type
List[str], optional

detector_type

Description of type of detector. Defaults to 'Pixel'.

Type
str, optional

mode

Acquisition mode for Tristan, either images or events. Defaults to events.

Type
str

Properties:
constants (Dict): Detector specific constants, such as locations of pixel_mask and flatfield files, and detector tick, frequency and timeslice rollover for event mode.

exception nexgen.nxs_utils.detector.**UnknownDetectorTypeError**

Bases: Exception

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

3.1.5 Source

Object definition for Source, Beam and Attenuator

class nexgen.nxs_utils.source.**Attenuator**(*transmission: float*)

Bases: DataClassJsonMixin

Attenuator definition.

class nexgen.nxs_utils.source.**Beam**(*wavelength: float*, *flux: float | None = None*)

Bases: DataClassJsonMixin

Beam definition.

class nexgen.nxs_utils.source.**Facility**(*name*, *short_name*, *type*, *id*)

Bases: tuple

Facility description

id

Alias for field number 3

name

Alias for field number 0

short_name

Alias for field number 1

type

Alias for field number 2

class nexgen.nxs_utils.source.**Source**(*beamline: str*, *facility: Facility = ('Diamond Light Source', 'DLS', 'Synchrotron X-ray Source', None)*, *probe: str | None = None*)

Bases: object

Source definition.

set_instrument_name() → str

Set the instrument name from the details saved in source.

If source type is not defined a priori, the function will assume it is a Synchrotron. If the facility_id is defined inside the source dictionary, that is the value that will be used. Naming tries to follow the recommended convention for NXmx: https://mmcif.wwpdb.org/dictionaries/mmcif_pdbx_v50.dic/Items/_diffrn_source.type.html

Returns
The name to write under '/entry/instrument/name'

Return type
name (str)

to_dict()

Write source information to a dictionary.

3.1.6 Sample

Sample definition utilities.

class nexgen.nxs_utils.sample.**Sample**(*name: 'str | None' = None*, *depends_on: 'str | None' = None*, *temperature: 'str | None' = None*)

Bases: `DataClassJsonMixin`

3.2 Writing tools

3.2.1 NXmx writers

For a standard NXmx data collection

class nexgen.nxs_write.nxmx_writer.**NXmxFileWriter**(*filename: Path | str*, *goniometer:* Goniometer, *detector:* Detector, *source:* Source, *beam:* Beam, *attenuator:* Attenuator, *tot_num_imgs: int*)

Bases: `object`

A class to generate NXmx format NeXus files.

add_NXnote(*notes: Dict*, *loc: str = '/entry/notes'*)

Save any additional information as NXnote at the end of the collection.

Parameters

- **notes** (*Dict*) – Dictionary of (key, value) pairs where key represents the dataset name and value its data.
- **loc** (*str, optional*) – Location in the NeXus file to save metadata. Defaults to "/entry/notes".

update_timestamps(*timestamp: datetime | str*, *dset_name: Literal['start_time', 'end_time', 'end_time_estimated'] = 'end_time'*)

Save timestamps for start and/or end collection if not written before.

Parameters

- **timestamp** (*datetime | str*) – Timestamp, as datetime or str.
- **dset_name** (*TSdset, optional*) – Name of dataset to write to nexus file. Allowed values: ["start_time", "end_time", "end_time_estimated". Defaults to "end_time".

write(*image_datafiles: List | None = None*, *image_filename: str | None = None*, *start_time: datetime | str | None = None*, *est_end_time: datetime | str | None = None*, *write_mode: str = 'x'*, *add_non_standard: bool = True*)

Write the NXmx format NeXus file.

This function calls the writers for the main NXclass objects.

Parameters

- **image_datafiles** (*List | None, optional*) – List of image data files. If not passed, the program will look for files with the stem_######.h5 in the target directory. Defaults to None.
- **image_filename** (*str | None, optional*) – Filename stem to use to look for image files. Needed in case it doesn't match the NeXus file name. Format: filename_runnumber. Defaults to None.

- **start_time** (*datetime | str, optional*) – Collection start time if available, in the format "%Y-%m-%dT%H:%M:%SZ".Defaults to None.
- **est_end_time** (*datetime | str, optional*) – Collection estimated end time if available, in the format "%Y-%m-%dT%H:%M:%SZ". Defaults to None.
- **write_mode** (*str, optional*) – String indicating writing mode for the output NeXus file. Accepts any valid h5py file opening mode. Defaults to "x".
- **add_non_standard** (*bool, optional*) – Flag if non-standard NXsample fields should be added for processing to work. Defaults to True, will change in the future.

write_vds(*vds_offset: int = 0*, *vds_shape: ~typing.Tuple[int, int, int] | None = None*, *vds_dtype: ~numpy.dtype[~typing.Any] | None | type[~typing.Any] | ~numpy._typing._dtype_like._SupportsDType[~numpy.dtype[~typing.Any]] | str | tuple[~typing.Any, int] | tuple[~typing.Any, ~typing.SupportsIndex | ~collections.abc.Sequence[~typing.SupportsIndex]] | list[~typing.Any] | ~numpy._typing._dtype_like._DTypeDict | tuple[~typing.Any, ~typing.Any] = <class 'numpy.uint16'>*, *clean_up: bool = False*)

Write a Virtual Dataset.

This method adds a VDS under /entry/data/data in the NeXus file, linking to either the full datasets or the subset defined by vds_offset (used as start index) and vds_shape. WARNING. Only use clean up if the data collection is finished and all the files have already been written.

Parameters

- **vds_offset** (*int, optional*) – Start index for the vds writer. Defaults to 0.
- **vds_shape** (*Tuple[int,int,int], optional*) – Shape of the data which will be linked in the VDS. If not passed, it will be defined as (tot_num_imgs - start_idx, *image_size). Defaults to None.
- **vds_dtype** (*DTypeLike, optional*) – The type of the input data. Defaults to np.uint16.
- **clean_up** (*bool, optional*) – Clean up unused links in vds. Defaults to False.

For an event-mode data collection using a Tristan detector

class nexgen.nxs_write.nxmx_writer.**EventNXmxFileWriter**(*filename: Path | str*, *goniometer: Goniometer*, *detector: Detector*, *source: Source*, *beam: Beam*, *attenuator: Attenuator*, *axis_end_position: float | None = None*)

Bases: *NXmxFileWriter*

A class to generate NXmx-like NeXus files for event mode data.

write(*start_time: datetime | str | None = None*, *write_mode: str = 'x'*, *add_non_standard: bool = False*)

Write a NXmx-like NeXus file for event mode data collections.

This method overrides the write() method of NXmxFileWriter, from which thsi class inherits.

Parameters

- **start_time** (*datetime | str, optional*) – Collection estimated end time if available, in the format "%Y-%m-%dT%H:%M:%SZ". Defaults to None.
- **write_mode** (*str, optional*) – String indicating writing mode for the output NeXus file. Accepts any valid h5py file opening mode. Defaults to "x".
- **add_non_standard** (*bool, optional*) – Flag if non-standard NXsample fields should be added for processing to work. Defaults to False.

For an Electron Diffraction collection using NXmx-like format nexus files. When dealing with an Electron Diffraction dataset, there may also be a need to convert the vectors to mcstas from another coordinate system convention, as well as save the relevant information about the new coordinate system into a NXcoordinate_system_set base class. This writer takes care of these issues.

class nexgen.nxs_write.nxmx_writer.**EDNXmxFileWriter**(*filename: Path | str*, *goniometer:* Goniometer, *detector:* Detector, *source:* Source, *beam:* Beam, *attenuator:* Attenuator, *tot_num_imgs: int*, *ED_coord_system: Dict*, *convert_to_mcstas: bool = False*)

Bases: *NXmxFileWriter*

A class to generate NXmx-like NeXus files for electron diffraction.

Requires an additional argument:
ED_coord_system (Dict): Definition of the current coordinate frame for ED. It should at least contain the convention, origin and base vectors.

add_NXnote(*notes: Dict*, *loc: str = '/entry/notes'*)

Save any additional information as NXnote at the end of the collection.

Parameters

- **notes** (*Dict*) – Dictionary of (key, value) pairs where key represents the dataset name and value its data.
- **loc** (*str, optional*) – Location in the NeXus file to save metadata. Defaults to "/entry/notes".

update_timestamps(*timestamp: datetime | str*, *dset_name: Literal['start_time', 'end_time', 'end_time_estimated'] = 'end_time'*)

Save timestamps for start and/or end collection if not written before.

Parameters

- **timestamp** (*datetime | str*) – Timestamp, as datetime or str.
- **dset_name** (*TSdset, optional*) – Name of dataset to write to nexus file. Allowed values: ["start_time", "end_time", "end_time_estimated". Defaults to "end_time".

write(*image_datafiles: List | None = None*, *data_entry_key: str = '/entry/data/data'*, *start_time: datetime | str | None = None*, *write_mode: str = 'x'*)

Write a NXmx-like NeXus file for electron diffraction.

This method overrides the write() method of NXmxFileWriter, from which thsi class inherits. In particular, it performs a few checks on the coordinate frame of the input vectors and then calls the writers for the relevant NeXus base classes.

Parameters

- **image_datafiles** (*List | None, optional*) – List of image data files. If not passed, the program will look for files with the stem_data_######.h5 in the target directory. Defaults to None.
- **data_entry_key** (*str, optional*) – Dataset entry key in datafiles. Defaults to entry/data/data.
- **start_time** (*datetime | str, optional*) – Collection estimated end time if available, in the format "%Y-%m-%dT%H:%M:%SZ". Defaults to None.
- **write_mode** (*str, optional*) – String indicating writing mode for the output NeXus file. Accepts any valid h5py file opening mode. Defaults to "x".

write_vds(*vds_dtype: ~numpy.dtype[~typing.Any] | None | type[~typing.Any] | ~numpy._typing._dtype_like._SupportsDType[~numpy.dtype[~typing.Any]] | str | tuple[~typing.Any, int] | tuple[~typing.Any, ~typing.SupportsIndex | ~collections.abc.Sequence[~typing.SupportsIndex]] | list[~typing.Any] | ~numpy._typing._dtype_like._DTypeDict | tuple[~typing.Any, ~typing.Any] = <class 'numpy.uint16'>, writer_type: str = 'dataset', data_entry_key: str = '/entry/data/data', datafiles: ~typing.List[~pathlib.Path] | None = None*)

Write a vds for electron diffraction.

This method overrides the write_vds() method of NXmxFileWriter, from which thsi class inherits. In particular, if required it will write an external vds file instead of a dataset.

Parameters

- **vds_dtype** (*DTypeLike, optional*) – The type of the input data. Defaults to np.uint16.
- **writer_type** (*str, optional*) – Type of vds required. Defaults to "dataset".
- **data_entry_key** (*str, optional*) – Dataset entry key in datafiles. Defaults to "/entry/data/data".
- **datafiles** ((*List | None, optional*) – List of image data files. If not passed, the program will look for files with the stem_######.h5 in the target directory. Defaults to None.

3.2.2 NXclass writers

All the NXclass writers available can be found in:

3.2.3 Old tools

Note: Tools such as ScanReader and write_nexus_from_scope have been deprecated as of version *0.8.0*. The functionality of *call_writers* has also been changed.

nexgen.command_line.cli_utils.**call_writers**(*nxsfile: Path | str, datafiles: List[Path | str], coordinate_frame: str, data_type: Tuple[str, int], goniometer: Dict[str, Any], detector: Dict[str, Any], module: Dict[str, Any], source: Dict[str, Any], beam: Dict[str, Any], attenuator: Dict[str, Any], metafile: bool = False, timestamps: Tuple[str, str] | None = None, notes: Dict[str, Any] | None = None*)

Call the writers for the NeXus base classes.

Parameters

- **nxsfile** (*Path | str*) – NeXus file to be written.
- **datafiles** (*List[Path | str]*) – List of at least 1 Path object to a HDF5 data file.
- **coordinate_frame** (*str*) – Coordinate system being used. Accepted frames are imgcif and mcstas.
- **data_type** (*Tuple[str, int]*) – Images or event-mode data, and eventually how many are being written.
- **(Dict[str** (*goniometer*)

- **description.** (*Any] Goniometer geometry*)
- **detector** (*Dict[str, Any]*) – Detector specific parameters and its axes.
- **module** (*Dict[str, Any]*) – Geometry and description of detector module.
- **source** (*Dict[str, Any]*) – Facility information.
- **beam** (*Dict[str, Any]*) – Beam properties.
- **attenuator** (*Dict[str, Any]*) – Attenuator properties.
- **metafile** (*bool, optional*) – Whether a metafile is present. Defaults to False.
- **timestamps** (*Tuple[str], optional*) – Start and end collection timestamps in ISO format. Defaults to None.
- **notes** (*Dict, optional*) – Any additional information to write as NXnote. Defaults to None.

3.2.4 Writing blank datasets

Generating blank images

Using an *Eiger* or *Tristan* detector mask …

nexgen.tools.data_writer.**build_an_eiger**(*image_size: List | Tuple*, *det_description: str*, *n_modules: Tuple[int, int] | None = None*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Generate an Eiger-like blank image.

Parameters

- **image_size** (*List | Tuple*) – Detector size, defines image dimensions as (slow_axis , fast_axis).
- **det_description** (*str*) – Identifies the type of Eiger detector.
- **n_modules** (*Tuple[int, int], optional*) – Number of modules in the detector. Defaults to None.

Returns
Blank image - an array of zeros with an Eiger-like mask.

Return type
IM (ArrayLike)

nexgen.tools.data_writer.**build_a_tristan**(*image_size: List | Tuple*, *det_description: str*) → _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]

Generate a Tristan-like blank image.

Parameters

- **image_size** (*List | Tuple*) – Detector size, fefines image dimensions as (slow_axis , fast_axis).
- **det_description** (*str*) – Identifies the Tristan detector.

Returns
Blank image - an array of zeros with an Eiger-like mask.

Return type
IM (ArrayLike)

nexgen.tools.data_writer.**generate_image_files**(*datafiles: List[Path | str]*, *image_size: List | Tuple*, *det_description: str*, *tot_num_images: int*)

Generate HDF5 files of blank images.

Parameters

- **datafiles** (*List[Path | str]*) – List of HDF5 files to be written.
- **image_size** (*List | Tuple*) – Image dimensions as (slow_axis, fast_axis).
- **det_description** (*str*) – Type of detector. The string should include the number of modules.
- **tot_num_images** (*int*) – Total number of images to be written across the files.

Raises
ValueError – If the number of files requested and the total number of images to write don't match.

Generating pseudo-events

nexgen.tools.data_writer.**pseudo_event_list**(*x_lim: Tuple[int, int | None]*, *y_lim: Tuple[int, int | None]*, *exp_time: float*) → Tuple[List, List]

Generate a pseudo-events list with positions and timestamps.

Parameters

- **x_lim** (*Tuple[int, Union[int, None]]*) – Minimum and maximum position along the fast axis.
- **y_lim** (*Tuple[int, Union[int, None]]*) – Minimum and maximum position along the slow axis.
- **exp_time** (*float*) – Total exposure time, in seconds.

Returns
Lists of pseudo-event positions and relative timestamps.

Return type
pos_list, time_list (Tuple[List, List])

nexgen.tools.data_writer.**generate_event_files**(*datafiles: List[Path | str]*, *num_chunks: int*, *det_description: str*, *exp_time: float*)

Generate HDF5 files of pseudo events.

Parameters

- **datafiles** (*List[Union[Path, str]]*) – List of HDF5 files to be written.
- **num_chunks** (*int*) – Chunks of events to be written per file.
- **det_description** (*str*) – Type of detector. The string should include the number of modules.
- **exp_time** (*float*) – Total exposure time, in seconds.

3.2.5 VDS writer

Tools to write Virtual DataSets

class nexgen.tools.vds_tools.**Dataset**(*name: 'str'*, *source_shape: 'Tuple[int]'*, *start_index: 'int' = 0*, *stop_index: 'int' = 1000*, *dest_shape: 'Tuple[int]' = None*)

nexgen.tools.vds_tools.**clean_unused_links**(*nxsfile: File*, *vds_shape: Tuple | List*, *start_index: int = 0*)

Remove links to external data not used in VDS.

Parameters

- **nxsfile** (*h5py.File*) – Handle to NeXus file being written.
- **vds_shape** (*Tuple | List*) – Actual shape of the VDS dataset, usually defined as (num_frames, *image_size).
- **start_index** (*int*) – The start point for the source data. Defaults to 0.

nexgen.tools.vds_tools.**create_virtual_layout**(*datasets: List[Dataset]*, *data_type: dtype[Any] | None | type[Any] | _SupportsDType[dtype[Any]] | str | tuple[Any, int] | tuple[Any, SupportsIndex | Sequence[SupportsIndex]] | list[Any] | _DTypeDict | tuple[Any, Any]*)

Create a virtual layout and populate it based on the provided data.

Parameters

- **datasets** (*List[Dataset]*) – A list of datasets that are to be merged.
- **data_type** (*DTypeLike*) – The type of the input data.

Returns
Virtual layout.

Return type
layout (h5py.VirtualLayout)

nexgen.tools.vds_tools.**find_datasets_in_file**(*nxdata: Group*) → List

Look for the source datasets in the NeXus file. Assumes that the source datasets are always h5py.ExternalLink.

Parameters
nxdata (*h5py.Group*) – Group where the data should be linked.

Raises
KeyError – If no ExternalLinks to data are found in the group.

Returns
The source datasets.

Return type
dsets (List)

nexgen.tools.vds_tools.**image_vds_writer**(*nxsfile: ~h5py._hl.files.File*, *full_data_shape: ~typing.Tuple | ~typing.List*, *start_index: int = 0*, *vds_shape: ~typing.Tuple | ~typing.List | None = None*, *data_type: ~numpy.dtype[~typing.Any] | None | type[~typing.Any] | ~numpy._typing._dtype_like._SupportsDType[~numpy.dtype[~typing.Any]] | str | tuple[~typing.Any, int] | tuple[~typing.Any, ~typing.SupportsIndex | ~collections.abc.Sequence[~typing.SupportsIndex]] | list[~typing.Any] | ~numpy._typing._dtype_like._DTypeDict | tuple[~typing.Any, ~typing.Any] = <class 'numpy.uint16'>*, *entry_key: str = 'data'*)

Virtual DataSet writer function for image data.

Parameters

- **nxsfile** (*h5py.File*) – Handle to NeXus file being written.
- **full_data_shape** (*Tuple | List*) – Shape of the full dataset, usually defined as (num_frames, *image_size).
- **start_index** (*int*) – The start point for the source data. Defaults to 0.
- **vds_shape** (*Tuple, optional*) – Desired shape of the VDS, usually defined as (num_frames, *image_size). The number of frames must be smaller or equal to the one in full_data_shape. Defaults to None.
- **data_type** (*DTypeLike, optional*) – The type of the input data. Defaults to np.uint16.
- **entry_key** (*str, optional*) – Entry key for the Virtual DataSet name. Defaults to data.

nexgen.tools.vds_tools.**jungfrau_vds_writer**(*nxsfile: ~h5py._hl.files.File*, *vds_shape: ~typing.Tuple | ~typing.List*, *data_type: ~numpy.dtype[~typing.Any] | None | type[~typing.Any] | ~numpy._typing._dtype_like._SupportsDType[~numpy.dtype[~typing.Any]] | str | tuple[~typing.Any, int] | tuple[~typing.Any, ~typing.SupportsIndex | ~collections.abc.Sequence[~typing.SupportsIndex]] | list[~typing.Any] | ~numpy._typing._dtype_like._DTypeDict | tuple[~typing.Any, ~typing.Any] = <class 'numpy.uint16'>*, *source_dsets: ~typing.List[str] | None = None*)

Write VDS for Jungfrau 1M use case, with a tiled layout.

nexgen.tools.vds_tools.**split_datasets**(*dsets*, *data_shape: Tuple[int, int, int]*, *start_idx: int = 0*, *vds_shape: Tuple[int, int, int] | None = None*) → List[*Dataset*]

Splits the full data shape and start index up into values per dataset, given that each dataset has a maximum size.

Parameters

- **dsets** (Dataset) – The input datasets.
- **data_shape** (*Tuple[int, int, int]*) – Shape of the data, usually defined as (num_frames, *image_size).
- **start_idx** (*int, optional*) – The start point for the source data. Defaults to 0.
- **vds_shape** (*Tuple, optional*) – Desired shape of the VDS, usually defined as (num_frames, *image_size). The number of frames must be smaller or equal to the one in full_data_shape. Defaults to None.

Raises

- **ValueError** – If the passed start index value is higher than the dataset lenght.
- **ValueError** – It the passed start index value is negative.

Returns
A list of datasets.

Return type
List[*Dataset*]

nexgen.tools.vds_tools.**vds_file_writer**(*nxsfile: ~h5py._hl.files.File, datafiles: ~typing.List[~pathlib.Path], data_shape: ~typing.Tuple | ~typing.List, data_type: ~numpy.dtype[~typing.Any] | None | type[~typing.Any] | ~numpy._typing._dtype_like._SupportsDType[~numpy.dtype[~typing.Any]] | str | tuple[~typing.Any, int] | tuple[~typing.Any, ~typing.SupportsIndex | ~collections.abc.Sequence[~typing.SupportsIndex]] | list[~typing.Any] | ~numpy._typing._dtype_like._DTypeDict | tuple[~typing.Any, ~typing.Any] = <class 'numpy.uint16'>, entry_key: str = 'data'*)

Write a Virtual DataSet _vds.h5 file for image data.

Parameters

- **nxsfile** (*h5py.File*) – NeXus file being written.
- **datafiles** (*List[Path]*) – List of paths to source files.
- **data_shape** (*Tuple | List*) – Shape of the dataset, usually defined as (num_frames, *image_size).
- **data_type** (*DTypeLike, optional*) – Dtype. Defaults to np.uint16.
- **entry_key** (*str*) – Entry key for the Virtual DataSet name. Defaults to data.

3.3 Copying tools

General tools to copy metadata from NeXus files.

nexgen.nxs_copy.copy_nexus.**images_nexus**(*data_file: List[Path | str], original_nexus: Path | str, simple_copy: bool = True, skip_group: List[str] = ['NXdata']*) → str

Copy NeXus metadata for images.

Parameters

- **data_file** (*List[Path | str]*) – HDF5 file with images.
- **original_nexus** (*Path | str*) – Original NeXus file with experiment metadata.
- **simple_copy** (*bool, optional*) – Copy everything from the original NeXus file. Defaults to True.
- **skip_group** (*List[str], optional*) – If simple_copy is False, list of NX_class objects to skip when copying. Defaults to [“NXdata”].

Returns
Filename of new NeXus file.

Return type
nxs_filename (str)

nexgen.nxs_copy.copy_nexus.**pseudo_events_nexus**(*data_file: List[Path | str]*, *original_nexus: Path | str*) → str

Copy NeXus metadata for pseudo event mode data.

Parameters

- **data_file** (*List[Path | str]*) – HDF5 with pseud event data.
- **original_nexus** (*Path | str*) – Original NeXus file with experiment metadata.

Returns
Filename of new NeXus file.

Return type
nxs_filename (str)

Tools for copying the metadata from Tristan NeXus files.

nexgen.nxs_copy.copy_tristan_nexus.**multiple_images_nexus**(*data_file: Path | str*, *tristan_nexus: Path | str*, *write_mode: str = 'x'*, *osc: float | None = None*, *nbins: int | None = None*) → str

Create a NeXus file for a multiple-image dataset or multiple image sequences from a pump-probe collection.

Copy the nexus tree from the original NeXus file for a collection on Tristan detector. There are two main applications for this function. In the first case, multiple images from a rotation collection have been binned and the scan_axis to be found in the input file is a (start, stop) tuple. The scan_axis in the new file will therefore be a list of angles. Osc and num_bins are mutually exclusive arguments to work out the scan_axis list. In the second case, multiple images from a 2D grid scan collection have been binned. The "rotation" scan_axis is still to be found in the input file as a (start, stop) tuple - although in this instance the two values should coincide. The values for the "translation" scan axes instead can be worked out from the chipmap dictionary, saved as a Unicode string inside the original NeXus file during collection, and nbins. It should be noted that passing osc in this case will raise an error and exit.

Parameters

- **data_file** (*Path | str*) – String or Path pointing to the HDF5 file containing the newly binned images.
- **tristan_nexus** (*Path | str*) – String or Path pointing to the input NeXus file with experiment metadata to be copied.
- **write_mode** (*str, optional*) – String indicating writing mode for the output NeXus file. Accepts any valid h5py file opening mode. Defaults to "x".
- **osc** (*float, optional*) – Oscillation angle (degrees). Defaults to None.
- **nbins** (*int, optional*) – Number of binned images. Defaults to None.

Raises

- **ValueError** – When osc has been passed instead of nbins for a grid scan collection.
- **ValueError** – When both osc and nbins have been passed for a rotation collection. The two values are mutually exclusive.
- **ValueError** – When neither osc nor nbins has been passed. It won't be possible to calculate the scan range without at least one of them.

Returns
The name of the output NeXus file.

Return type
nxs_filename (str)

nexgen.nxs_copy.copy_tristan_nexus.**serial_images_nexus**(*data_file: Path | str*, *tristan_nexus: Path | str*, *nbins: int*, *write_mode: str = 'x'*) → str

Create a NeXus file for a serial collection.

Parameters

- **data_file** (*Path | str*) – String or Path pointing to the HDF5 file containing the newly binned images.
- **tristan_nexus** (*Path | str*) – String or Path pointing to the input NeXus file with experiment metadata to be copied.
- **nbins** (*int*) – Number of binned images.
- **write_mode** (*str, optional*) – String indicating writing mode for the output NeXus file. Accepts any valid h5py file opening mode. Defaults to "x".

Returns
description

Return type
str

nexgen.nxs_copy.copy_tristan_nexus.**single_image_nexus**(*data_file: Path | str*, *tristan_nexus: Path | str*, *write_mode: str = 'x'*, *pump_probe_bins: int | None = None*) → str

Create a NeXus file for a single-image or a stationary pump-probe dataset.

Copy the nexus tree from the original NeXus file for a collection on Tristan detector. In the case of a single image, the input scan_axis is a (start, stop) tuple where start and stop have the same value, for a pump-probe experiment the values might differ for some older datasets. The scan_axis in the new file will therefore be one single number, equal to the "start".

Parameters

- **data_file** (*Path | str*) – String or Path pointing to the HDF5 file containing the newly binned images.
- **tristan_nexus** (*Path | str*) – String or Path pointing to the input NeXus file with experiment metadata to be copied.
- **write_mode** (*str, optional*) – String indicating writing mode for the output NeXus file. Accepts any valid h5py file opening mode. Defaults to "x".
- **pump_probe_bins** (*int, optional*) – If the NeXus file is be linked to a static pump-probe image stack, pass the number of images the events have been binned into. Deafults to None.

Returns
The name of the output NeXus file.

Return type
nxs_filename (str)

3.4 Utilities

General utilities for nexgen

nexgen.utils.**get_filename_template**(*input_filename: Path*) → str

Get the data file name template from either the master or the meta file.

Parameters
input_filename (*Path*) – Path object containing the name of master or meta file. The format should be either file_master.h5, file.nxs for a master file, file_meta.h5 for a meta file.

Raises
NameError – If the input file does not have the expected format.

Returns
String template for the name of blank data file.

Return type
filename_template (str)

nexgen.utils.**get_iso_timestamp**(*ts: str | float*) → str

Format a timestamp string to be stores in a NeXus file according to ISO8601: 'YY-MM-DDThh:mm:ssZ'

Parameters
ts (*str | float*) – Input string, can also be a timestamp (eg. time.time()) string. Allowed formats: "%Y-%m-%dT%H:%M:%S", "%Y-%m-%d %H:%M:%S", "%a %b %d %Y %H:%M:%S", "%A, %d. %B %Y %I:%M%p".

Returns
Formatted timestamp.

Return type
ts_iso (str)

nexgen.utils.**get_nexus_filename**(*input_filename: Path | str*, *copy: bool = False*) → Path

Get the filename for the NeXus file from the stem of the input file name.

Parameters

- **input_filename** (*Path | str*) – File name and path of either a .h5 data file or a _meta.h5 file.
- **copy** (*bool, optional*) – Avoid trying to write a new file with the same name as the old one when making a copy. Defaults to False.

Returns
NeXus file name (.nxs) path.

Return type
Path

nexgen.utils.**units_of_length**(*q: str | float*, *to_base: bool = False*) → Quantity

Check that a quantity of length is compatible with NX_LENGTH, defaulting to m if dimensionless.

Parameters

- **q** (*Any*) – An object that can be interpreted as a pint Quantity, it can be dimensionless.
- **to_base** (*bool, optional*) – If True, convert to base units of length (m). Defaults to False.

Raises

- **ValueError** – If the input value is a negative number.
- **pint.errors.DimensionalityError** – If the input value is not a quantity of lenght.

Returns
A pint quantity with units applied if it was dimensionless.

Return type
quantity (pint.Quantity)

nexgen.utils.**units_of_time**(*q: str*) → Quantity

Check that a quantity of time is compatible with NX_TIME, defaulting to s if dimensionless. Convert to seconds if time is passed as a fraction of it.

Parameters
q (*str*) – A string that can be interpreted as a pint Quantity, it can be dimensionless.

Raises

- **ValueError** – If the input value is a negative number.
- **pint.errors.DimensionalityError** – If the input value is not a quantity of lenght.

Returns
A pint quantity in s, with units applied if it was dimensionless.

Return type
quantity (pint.Quantity)

nexgen.utils.**walk_nxs**(*nxs_obj: File | Group*) → List[str]

Walk all the groups, subgroups and datasets of an object.

Parameters
nxs_obj (*h5py.File | h5py.Group*) – Object to walk through, could be a file or a group.

Returns
List of objects found, as strings.

Return type
obj_list (List[str])

Writing tools Utilities for writing new NeXus format files.

nexgen.nxs_write.write_utils.**add_sample_axis_groups**(*nxsample: Group*, *axis_list: List[Axis]*)

Add non-standard "sample_{phi,omega,...}" groups to NXsample. These may be needed for some autoprocessing tools to work correctly.

Parameters

- **nxsample** (*h5py.Group*) – NeXus NXsample group.
- **axis_list** (*List[Axis]*) – List of goniometer axes.

nexgen.nxs_write.write_utils.**calculate_origin**(*beam_center_fs: List | Tuple*, *fs_pixel_size: List | Tuple*, *fast_axis_vector: Tuple*, *slow_axis_vector: Tuple*, *mode: str = '1'*) → Tuple[List, float]

Calculate the offset of the detector.

This function returns the detector origin array, which is saved as the vector attribute of the module_offset field. The value to set the module_offset to is also returned: the magnitude of the displacement if the vector is normalized, 1.0 otherwise Assumes that fast and slow axis vectors have already been converted to mcstas if needed.

Parameters

- **beam_center_fs** (*List | Tuple*) – Beam center position in fast and slow direction.
- **fs_pixel_size** (*List | Tuple*) – Pixel size in fast and slow direction, in m.
- **fast_axis_vector** (*Tuple*) – Fast axis vector.
- **slow_axis_vector** (*Tuple*) – Slow axis vector.
- **mode** (*str, optional*) – Decide how origin should be calculated. If set to "1" the displacement vector is un-normalized and the offset value set to 1.0. If set to "2" the displacement is normalized and the offset value is set to the magnitude of the displacement. Defaults to "1".

Returns
Displacement of beam center, vector attribute of module_offset. offset_val (float): Value to assign to module_offset, depending whether det_origin is normalized or not.

Return type
det_origin (List)

nexgen.nxs_write.write_utils.**create_attributes**(*nxs_obj: Group | Dataset*, *names: Tuple*, *values: Tuple*)

Create or overwrite attributes with additional metadata information.

Parameters

- **nxs_obj** (*h5py.Group | h5py.Dataset*) – NeXus object to which the attributes should be attached.
- **names** (*Tuple*) – The names of the new attributes.
- **values** (*Tuple*) – The attribute values asociated to the names.

nexgen.nxs_write.write_utils.**find_number_of_images**(*datafile_list: List[Path]*, *entry_key: str = 'data'*) → int

Calculate total number of images when there's more than one input HDF5 file.

Parameters

- **datafile_list** (*List[Path]*) – List of paths to the input image files.
- **entry_key** (*str*) – Key for the location of the images inside the data files. Defaults to "data".

Returns
Total number of images.

Return type
num_images (int)

nexgen.nxs_write.write_utils.**mask_and_flatfield_writer**(*nxdet_grp: Group*, *dset_name: str*, *dset_data: str | _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*, *applied_val: bool*)

Utility function to write mask or flatfield to NXdetector group for image data when not already linked to the _meta.h5 file. If the pixel_mask/flatfield data is passed as a string, it will be assumed to be a file path and the writer will try to set up an external link to it.

Parameters

- **nxdet_grp** (*h5py.Group*) – Handle to HDF5 NXdetector group.

- **dset_name** (*str*) – Name of the new field/dataset to be written.
- **dset_data** (*str | ArrayLike*) – Dataset data to be written in the field. Can be a string or an array-like dataset. If the data type is a numpy ndarray, it will be compressed before writing.
- **applied_val** (*bool*) – Value to write to *{flatfield,pixel_mask}_applied* fields.

nexgen.nxs_write.write_utils.**set_dependency**(*dep_info: str*, *path: str | None = None*)

Define value for "depends_on" attribute. If the attribute points to the head of the dependency chain, simply pass "." for dep_info.

Parameters

- **dep_info** (*str*) – The name of the transformation upon which the current one depends on.
- **path** (*str*) – Where the transformation is. Set to None, if passed it points to location in the NeXus tree.

Returns

The value to be passed to the attribute "depends_on"

nexgen.nxs_write.write_utils.**write_compressed_copy**(*nxgroup: Group*, *dset_name: str*, *data: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | None = None*, *filename: Path | str | None = None*, *filter_choice: str = 'bitshuffle'*, *dset_key: str = 'image'*, ***kwargs*)

Write a compressed copy of some dataset in the desired HDF5 group, using the filter of choice with lz4 compression. Available filters at this time include "Blosc" and "Bitshuffle" (default). The main application for this function in nexgen is to write a compressed copy of a pixel mask or a flatfield file/dataset directly into the NXdetector group of a NXmx NeXus file. The data and filename arguments are mutually exclusive as only one of them can be used as input. If a filename is passed, it is also required to pass the key for the relevant dataset to be copied. Failure to do so will result in nothing being written to the NeXus file.

Parameters

- **nxgroup** (*h5py.Group*) – Handle to HDF5 group.
- **dset_name** (*str*) – Name of the new dataset to be written.
- **data** (*ArrayLike, optional*) – Dataset to be compressed. Defaults to None.
- **filename** (*Path | str, optional*) – Filename containing the dataset to be compressed into the NeXus file. Defaults to None.
- **filter_choice** (*str, optional*) – Filter to be used for compression. Either blosc or bitshuffle. Defaults to bitshuffle.
- **dset_key** (*str, optional*) – Dataset name inside the passed file. Defaults to "image".

Keyword Arguments

block_size (*int, optional*) – Number of elements per block, it needs to be divisible by 8. Needed for Bitshuffle filter. Defaults to 0.

Raises

ValueError – If both a dataset and a filename have been passed to the function.

Copying tools Utilities for copying metadata to new NeXus files.

nexgen.nxs_copy.copy_utils.**compute_ssx_axes**(*nxs_in: File*, *nbins: int*, *rot_ax: str*, *rot_val: Tuple | List | _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes]*) → Tuple[Dict, Dict, Dict, int | None]

Computes the positions on chip corresponding to the binned images from a Tristan fixed target collection.

The function looks for the blocks (chipmap) and the chip_info dictionaries inside the original NeXus file and calculates the scan points from there. For older versions of the SSX NeXus files, this information is not available and the scan points will be calculated based on the number of images and using the default Oxford chip dimensions, starting from the upper left corned of the chip. If multiple windows have been binned into a single image, instead of the sam_(x,y) translation axes values, the number of windows per images will be returned and saved in the NeXus file.

Parameters

- **nxs_in** (*h5py.File*) – File handle for the original Tristan collection NeXus file.
- **nbins** (*int*) – Number of images.
- **rot_ax** (*str*) – Rotation axis.
- **rot_val** (*Tuple | List | ArrayLike*) – Rotation axis start and stop values, as found in the original NeXus.

Returns

Oscillation range, Translation range, pump_probe info, number of windows per binned image.

Return type

OSC, TRANSL, pump_info, windows_per_bin (Tuple[Dict, Dict, Dict, int | None])

nexgen.nxs_copy.copy_utils.**convert_scan_axis**(*nxsample: Group*, *nxdata: Group*, *ax: str*, *ax_range: _SupportsArray[dtype[Any]] | _NestedSequence[_SupportsArray[dtype[Any]]] | bool | int | float | complex | str | bytes | _NestedSequence[bool | int | float | complex | str | bytes] | None = None*)

Modify all instances of scan_axis present in NeXus file NXsample group.

Parameters

- **nxsample** (*h5py.Group*) – NXsample group of NeXus file to be modified.
- **nxdata** (*h5py.Group*) – NXdata group of NeXus file to be modified.
- **ax** (*str*) – Name of scan_axis.
- **ax_range** (*ArrayLike*) – Scan points. If passed, axis_increment_set and axis_end will also be written. Defaults to None

nexgen.nxs_copy.copy_utils.**get_nexus_tree**(*nxs_in: File*, *nxs_out: File*, *skip: bool = True*, *skip_obj: List[str] | None = None*) → Group | None

Copy the tree from the original NeXus file. Everything except NXdata is copied to a new NeXus file. If skip is False, then the full tree is copied.

Parameters

- **nxs_in** (*h5py.File*) – Original NeXus file.
- **nxs_out** (*h5py.File*) – New NeXus file.
- **skip** (*bool, optional*) – Copy everything but objects in skip_obj, which always include NXdata. Pass False to copy the whole NXentry tree. Defaults to True.

- **skip_obj** (*List[str], optional*) – List of NX_class objects not to be copied, eg. 'NXdata' or 'NXdetector'.. Defaults to None.

Returns
The group NXentry or nothing if the full file is copied.

Return type
h5py.Group | None

nexgen.nxs_copy.copy_utils.**get_skip_list**(*nxentry: Group*, *skip_obj: List[str]*) → List[str]

Get a list of all the objects that should not be copied in the new NeXus file.

Parameters

- **nxentry** (*h5py.Group*) – NXentry group of a NeXus file.
- **skip_obj** (*List[str]*) – List of objects that should not be copied.

Returns
List of "NXclass" objects to skip during copy.

Return type
skip_list (List[str])

nexgen.nxs_copy.copy_utils.**h5str**(*h5_value: str | bytes_ | bytes*) → str

Convert a value returned an h5py attribute to str.

h5py can return either a bytes-like (**numpy.string_**) or str object for attribute values depending on whether the value was written as fixed or variable length. This function collapses the two to str.

Parameters
h5_value (str | **np.string_** | bytes) – Original attribute value.

Returns
Attribute value collapsed to str.

Return type
str

nexgen.nxs_copy.copy_utils.**identify_tristan_scan_axis**(*nxs_in: File*) → Tuple[str | None, Dict[str, Any]]

Identify the scan_axis in the NeXus tree of a Tristan collection.

Return the first data set in the group '/entry/data' that has the attribute 'transformation_type' equal to 'rotation'.

Parameters
nxs_in (*h5py.File*) – Tristan NeXus file

Returns
Name of the scan_axis. ax_attrs (Dict[str, Any]): Attributes of the scan_axis dataset.

Return type
ax (str | None)

nexgen.nxs_copy.copy_utils.**is_chipmap_in_tristan_nxs**(*nxobj: File | Group*, *loc: str = 'entry/source/notes/chipmap'*) → bool

Look for the saved chipmap for a SSX experiment inside a tristan nexus file.

Parameters

- **nxobj** (*h5py.File | h5py.Group*) – NeXus object to be searched, could be a file or a group.

- **loc** (*str, optional*) – Location where the chipmap should be saved. Defaults to "entry/source/notes/chipmap".

Returns
Returns True is a chipmap is found, False otherwise.

Return type
bool

3.4.1 HDF5 metafile reader

Metafile definition: Define a Metafile object to describe the _meta.h5 file and get the necessary information from it.

class nexgen.tools.metafile.**DectrisMetafile**(*handle: File*)

Bases: Metafile

Describes a _meta.h5 file for a Dectris Eiger detector.

class nexgen.tools.metafile.**TristanMetafile**(*handle: File*)

Bases: Metafile

Describes a _meta.h5 file for a Tristan detector.

When operating a Dectris detector, the goniometer and detector axes values are usually stored in the *config/* dataset.

nexgen.tools.meta_reader.**update_axes_from_meta**(*meta_file: DectrisMetafile*, *axes_list: List[Axis]*, *osc_axis: str | None = None*, *use_config: bool = False*)

Update goniometer or detector axes values from those stores in the _dectris group.

Parameters

- **meta_file** (DectrisMetafile) – Handle to Dectris-shaped meta.h5 file.
- **axes_list** (*List[Axis]*) – List of axes to look up and eventually update.
- **osc_axis** (*str | None, optional*) – If passed, the number of images corresponding to the osc_axis will be updated too. Defaults to None.
- **use_config** (*bool, optional*) – If passed read from config dataset in meta file instead of _dectris group. Defaults to False.

If there's a need to write a VDS dataset from data collected on a Dectris detector, it might be useful to first find out the data type using the information stored in the *meta* file.

nexgen.tools.meta_reader.**define_vds_data_type**(*meta_file: DectrisMetafile*) → dtype[Any] | None | type[Any] | _SupportsDType[dtype[Any]] | str | tuple[Any, int] | tuple[Any, SupportsIndex | Sequence[SupportsIndex]] | list[Any] | _DTypeDict | tuple[Any, Any]

Define the data type for the VDS from the bit_depth defined in the meta file.

Parameters
meta_file (DectrisMetafile) – Handle to Dectris-shaped meta.h5 file.

Returns
Data type as np.uint##.

Return type
DTypeLike

3.4.2 Reader for Singla detector master file

Tools to calculate the beam center of an Electron Diffraction experiment:

3.5 Logging configuration

Logging configuration.

class nexgen.log.**LoggingContext**(*logger*, *level=None*)

Define a basic context manager for selective logging. See https://docs.python.org/3/howto/logging-cookbook.html#using-a-context-manager-for-selective-logging.

nexgen.log.**config**(*logfile: str | None = None*, *write_mode: str = 'a'*, *delayed: bool = False*)

Configure the logging.

Parameters

- **logfile** (*str, optional*) – If passed, create a file handler for the logger to write to file the log output. Defaults to None.
- **write_mode** (*str, optional*) – String indicating writing mode for the output .log file. Defaults to "a".
- **delayed** (*bool, optional*) – Setting for the FileHandler delay option. If true, then file opening is deferred until the first call to *emit*.Defaults to False.

CHAPTER

FOUR

BEAMLINES API

4.1 General utilities

class nexgen.beamlines.beamline_utils.**BeamlineAxes**(*gonio: List[Axis]*, *det_axes: List[Axis]*, *fast_axis: Point3D | Tuple[float, float, float]*, *slow_axis: Point3D | Tuple[float, float, float]*)

Beamline specific axes for goniometer, detector and detector module.

class nexgen.beamlines.beamline_utils.**PumpProbe**(*pump_status: bool = False*, *pump_exposure: float | None = None*, *pump_delay: float | None = None*, *pump_repeat: int | None = 0*)

Define pump probe parameters.

Parameters

- **pump_status** (*bool*) – Pump on/off
- **pump_exposure** (*float, optional*) – Pump exposure time, in s.
- **pump_delay** (*float, optional*) – Pump delay, in s.
- **pump_repeat** (*int, optional*) – Repeat mode.

4.2 I19-2

1. Directly from the python intepreter/ a python script …

The function

nexgen.beamlines.I19_2_nxs.**nexus_writer**(*meta_file: Path | str*, *detector_name: str*, *exposure_time: float*, *scan_axis: str = 'phi'*, *start_time: datetime | None = None*, *stop_time: datetime | None = None*, ***params*)

Gather all parameters from the beamline and call the NeXus writers.

Parameters

- **meta_file** (*Path | str*) – Path to _meta.h5 file.
- **detector_name** (*str*) – Detector in use.
- **exposure_time** (*float*) – Exposure time, in s.
- **scan_axis** (*str, optional*) – Name of the oscillation axis. Defaults to phi.
- **start_time** (*datetime, optional*) – Experiment start time. Defaults to None.

- **stop_time** (*datetime, optional*) – Experiment end time. Defaults to None.

Keyword Arguments

- **n_imgs** (*int*) – Total number of images to be collected.
- **transmission** (*float*) – Attenuator transmission, in %.
- **wavelength** (*float*) – Wavelength of incident beam, in A.
- **beam_center** (*List[float, float]*) – Beam center position, in pixels.
- **gonio_pos** (*List[axes]*) – Name, start and end positions of the goniometer axes.
- **det_pos** (*List[det_axes]*) – Name, start and end positions of detector axes.
- **outdir** (*str*) – Directory where to save the file. Only specify if different from meta_file directory.
- **serial** (*bool*) – Specify whether it's a serial crystallography dataset.
- **det_dist** (*float*) – Distance between sample and detector, in mm.
- **use_meta** (*bool*) – For Eiger, if True use metadata from meta.h5 file. Otherwise will require all other information to be passed manually.

can be called from python and depending on the specified detector type will run:

nexgen.beamlines.I19_2_nxs.**tristan_writer**(*master_file: Path*, *TR:* CollectionParams, *timestamps: Tuple[str, str] = (None, None)*, *axes_pos: List[*axes*] | None = None*, *det_pos: List[*det_axes*] | None = None*)

A function to call the nexus writer for Tristan 10M detector.

Parameters

- **master_file** (*Path*) – Path to nexus file to be written.
- **TR** (CollectionParams) – Parameters passed from the beamline.
- **timestamps** (*Tuple[str, str], optional*) – Collection start and end time. Defaults to (None, None).
- **axes_pos** (*List[axes], optional*) – List of (axis_name, start, end) values for the goniometer, passed from command line. Defaults to None.
- **det_pos** (*List[det_axes], optional*) – List of (axis_name, start) values for the detector, passed from command line. Defaults to None.

nexgen.beamlines.I19_2_nxs.**eiger_writer**(*master_file: Path*, *TR:* CollectionParams, *timestamps: Tuple[str, str] = (None, None)*, *use_meta: bool = False*, *n_frames: int | None = None*, *axes_pos: List[*axes*] | None = None*, *det_pos: List[*det_axes*] | None = None*, *vds_offset: int = 0*)

A function to call the NXmx nexus file writer for Eiger 2X 4M detector. If use_meta is set to False, axes_pos and det_pos become required arguments. Otherwise, axes_pos and det_pos can be None but the code requires the information contained inside the meta file to work correctly.

Parameters

- **master_file** (*Path*) – Path to nexus file to be written.
- **TR** (CollectionParams) – Parameters passed from the beamline.
- **timestamps** (*Tuple[str, str], optional*) – Collection start and end time. Defaults to (None, None).

- **use_meta** (*bool, optional*) – If True, metadata such as axes positions, wavelength etc. will be updated using the meta.h5 file. Defaults to False.
- **num_frames** (*int, optional*) – Number of images for the nexus file. Not necessary if same as the tot_num_images from the CollectionParameters. If different, the VDS will onlu contain the number of frames specified here. Defaults to None.
- **axes_pos** (*List[axes], optional*) – List of (axis_name, start, inc) values for the goniometer, passed from command line. Defaults to None.
- **det_pos** (*List[det_axes], optional*) – List of (axis_name, start) values for the detector, passed from command line. Defaults to None.
- **vds_offset** (*int, optional*) – Start index for the vds writer. Defaults to 0.

Raises

- **ValueError** – If use_meta is set to False but axes_pos and det_pos haven't been passed.
- **IOError** – If the axes positions can't be read from the metafile (missing config or broken links).

Some useful type definitions to use with these methods:

class nexgen.beamlines.I19_2_nxs.**axes**(*id=None*, *start=0.0*, *inc=0.0*, *end=0.0*)

Goniometer axis name, start and end position, increment.

end

Axis end position. Defaults fo 0.0. Only really needed for Tristan collections.

id

Axis name.

inc

Axis increment value. Defaults fo 0.0. Only needed for the scan axis.

start

Axis start position. Defaults fo 0.0.

class nexgen.beamlines.I19_2_nxs.**det_axes**(*id=None*, *start=0.0*)

Detector axis name and position.

id

Axis name.

start

Axis position. Defaults to 0.0.

Collection parameters schema for I19-2

pydantic model nexgen.beamlines.I19_2_nxs.**CollectionParams**

Parameters passed as input from the beamline.

Parameters

- **metafile** – Path to _meta.h5 file.
- **detector_name** – Name of the detector in use for current experiment.
- **exposure_time** – Exposure time, in s.
- **beam_center** – Beam center (x,y) position, in pixels.
- **wavelength** – Incident beam wavelength, in A.

- **transmission** – Attenuator transmission, in %.
- **tot_num_images** – Total number of frames in a collection.
- **scan_axis** – Rotation scan axis. Must be passed for Tristan.


```
{
   "title": "CollectionParams",
   "description": "Parameters passed as input from the beamline.\n\nArgs:\n    
↪metafile: Path to _meta.h5 file.\n    detector_name: Name of the detector in use
↪for current experiment.\n    exposure_time: Exposure time, in s.\n    beam_
↪center: Beam center (x,y) position, in pixels.\n    wavelength: Incident beam
↪wavelength, in A.\n    transmission: Attenuator transmission, in %.\n    tot_num_
↪images: Total number of frames in a collection.\n    scan_axis: Rotation scan
↪axis. Must be passed for Tristan.",
   "type": "object",
   "properties": {
      "metafile": {
         "title": "Metafile",
         "anyOf": [
            {
               "type": "string",
               "format": "path"
            },
            {
               "type": "string"
            }
         ]
      },
      "detector_name": {
         "title": "Detector Name",
         "type": "string"
      },
      "exposure_time": {
         "title": "Exposure Time",
         "type": "number"
      },
      "beam_center": {
         "title": "Beam Center",
         "type": "array",
         "items": {
            "type": "number"
         }
      },
      "wavelength": {
         "title": "Wavelength",
         "type": "number"
      },
      "transmission": {
         "title": "Transmission",
         "type": "number"
      },
      "tot_num_images": {
         "title": "Tot Num Images",
```


(continues on next page)

(continued from previous page)


```
            "type": "integer"
        },
        "scan_axis": {
            "title": "Scan Axis",
            "type": "string"
        }
    },
    "required": [
        "metafile",
        "detector_name",
        "exposure_time",
        "beam_center"
    ]
}
```


Fields

- *beam_center (Sequence[float])*
- *detector_name (str)*
- *exposure_time (float)*
- *metafile (pathlib.Path | str)*
- *scan_axis (str | None)*
- *tot_num_images (int | None)*
- *transmission (float | None)*
- *wavelength (float | None)*

field beam_center: Sequence[float] [Required]

field detector_name: str [Required]

field exposure_time: float [Required]

field metafile: Path | str [Required]

field scan_axis: str | None = None

field tot_num_images: int | None = None

field transmission: float | None = None

field wavelength: float | None = None

2. Interface with GDA …

class nexgen.beamlines.I19_2_gda_nxs.**tr_collect**(*meta_file*, *xml_file*, *detector_name*, *exposure_time*, *wavelength*, *beam_center*, *start_time*, *stop_time*, *geometry_json*, *detector_json*)

Information extracted from GDA containing collection parameters.

beam_center

Beam center (x,y) position, in pixels.

detector_json
Path to GDA-generated JSON file describing the detector.

detector_name
Name of the detector in use for current experiment.

exposure_time
Exposure time, in s.

geometry_json
Path to GDA-generated JSON file describing the beamline geometry.

meta_file
Path to _meta.h5 file.

start_time
Collection start time.

stop_time
Collection end time.

wavelength
Incident beam wavelength, in A.

xml_file
Path to GDA-generated xml file.

nexgen.beamlines.I19_2_gda_nxs.**tristan_writer**(*master_file: Path*, *TR: namedtuple*, *axes_params:* BeamlineAxes, *det_params:* EigerDetector | TristanDetector | SinglaDetector | JungfrauDetector | CetaDetector, *timestamps: Tuple[str, str] = (None, None)*)

A function to call the nexus writer for Tristan 10M detector.

Parameters

- **master_file** (*Path*) – Path to nexus file to be written.
- **TR** (*namedtuple*) – Parameters passed from the beamline.
- **axes_params** (BeamlineAxes) – Axes for goniometer, detector and detector module.
- **det_params** (*DetectorType*) – Detector definition for Tristan.
- **timestamps** (*Tuple[str, str], optional*) – Collection start and end time. Defaults to None.

nexgen.beamlines.I19_2_gda_nxs.**eiger_writer**(*master_file: ~pathlib.Path*, *TR: ~collections.namedtuple*, *axes_params: ~nexgen.beamlines.beamline_utils.BeamlineAxes*, *det_params: ~nexgen.nxs_utils.detector.EigerDetector | ~nexgen.nxs_utils.detector.TristanDetector | ~nexgen.nxs_utils.detector.SinglaDetector | ~nexgen.nxs_utils.detector.JungfrauDetector | ~nexgen.nxs_utils.detector.CetaDetector*, *timestamps: ~typing.Tuple[str, str] = (None, None)*, *vds_dtype: ~numpy.dtype[~typing.Any] | None | type[~typing.Any] | ~numpy._typing._dtype_like._SupportsDType[~numpy.dtype[~typing.Any]] | str | tuple[~typing.Any, int] | tuple[~typing.Any, ~typing.SupportsIndex | ~collections.abc.Sequence[~typing.SupportsIndex]] | list[~typing.Any] | ~numpy._typing._dtype_like._DTypeDict | tuple[~typing.Any, ~typing.Any] = <class 'numpy.uint16'>*)

A function to call the nexus writer for Eiger 2X 4M detector.

Parameters

- **master_file** (*Path*) – Path to nexus file to be written.
- **TR** (*namedtuple*) – Parameters passed from the beamline.
- **axes_params** (BeamlineAxes) – Axes for goniometer, detector and detector module.
- **det_params** (*DetectorType*) – Detector definition for Eiger.
- **timestamps** (*Tuple[str, str], optional*) – Collection start and end time. Defaults to (None, None).
- **vds_dtype** (*DtypeLike*) – Data type for vds as np.uint##.

4.3 Serial crystallography: Eiger writers

nexgen.beamlines.SSX_Eiger_nxs.**ssx_eiger_writer**(*visitpath: Path | str*, *filename: str*, *beamline: str*, *num_imgs: int*, *expt_type: str = 'fixed-target'*, *pump_status: bool = False*, ***ssx_params*)

Gather all collection parameters and write the NeXus file for SSX using Eiger detector.

Parameters

- **visitpath** (*Path | str*) – Collection directory.
- **filename** (*str*) – Filename root.
- **beamline** (*str*) – Beamline on which the experiment is being run.
- **num_imgs** (*int*) – Total number of images collected.
- **expt_type** (*str, optional*) – Experiment type, accepted values: extruder, fixed-target, 3Dgridscan. Defaults to “fixed-target”.
- **pump_status** (*bool, optional*) – True for pump-probe experiment. Defaults to False.

Keyword Arguments

- **exp_time** (*float*) – Exposure time, in s.
- **det_dist** (*float*) – Distance between sample and detector, in mm.
- **beam_center** (*List[float, float]*) – Beam center position, in pixels.
- **transmission** (*float*) – Attenuator transmission, in %.
- **wavelength** (*float*) – Wavelength of incident beam, in A.
- **flux** (*float*) – Total flux.
- **start_time** (*datetime*) – Experiment start time.
- **stop_time** (*datetime*) – Experiment end time.
- **chip_info** (*Dict*) – For a grid scan, dictionary containing basic chip information. At least it should contain: x/y_start, x/y number of blocks and block size, x/y number of steps and number of exposures.
- **chipmap** (*Path | str*) – Path to the chipmap file corresponding to the experiment, if None for a fixed target experiment, it indicates that the fullchip is being scanned.
- **pump_exp** (*float*) – Pump exposure time, in s.
- **pump_delay** (*float*) – Pump delay time, in s.
- **osc_axis** (*str*) – Oscillation axis. Always omega on I24. If not passed it will default to phi for I19-2.
- **outdir** (*str*) – Directory where to save the file. Only specify if different from meta_file directory.

Raises

- **ValueError** – If an invalid beamline name is passed.
- **ValueError** – If an invalid experiment type is passed.

4.4 Serial crystallography: Tristan writers

nexgen.beamlines.SSX_Tristan_nxs.**ssx_tristan_writer**(*visitpath: Path | str*, *filename: str*, *beamline: str*, ***ssx_params*)

Gather all parameters from the beamline and call the NeXus writers.

Parameters

- **visitpath** (*Path | str*) – Path to colection directory.
- **filename** (*str*) – Root of the filename.
- **beamline** (*str*) – Beamline on which the experiment is being run.

Keyword Arguments

- **exp_time** (*float*) – Exposure time, in s.
- **det_dist** (*float*) – Distance between sample and detector, in mm.
- **beam_center** (*List[float, float]*) – Beam center position, in pixels.
- **transmission** (*float*) – Attenuator transmission, in %.
- **wavelength** (*float*) – Wavelength of incident beam, in A.

- **flux** (*float*) – Total flux.
- **start_time** (*datetime*) – Experiment start time.
- **stop_time** (*datetime*) – Experiment end time.
- **chip_info** (*Dict*) – For a grid scan, dictionary containing basic chip information. At least it should contain: x/y_start, x/y number of blocks and block size, x/y number of steps and number of exposures.
- **chipmap** (*Path | str*) – Path to the chipmap file corresponding to the experiment, or 'fullchip' indicating that the whole chip is being scanned.

4.5 Serial crystallography: chip tools

Tools to read a chip and compute the coordinates of a Serial Crystallography collection.

class nexgen.beamlines.SSX_chip.**Chip**(*name: str, num_steps: List[int, int] | Tuple[int, int], step_size: List[float, float] | Tuple[float, float], num_blocks: List[int, int] | Tuple[int, int], block_size: List[float, float] | Tuple[float, float], start_pos: List[float, float, float] = <factory>*)

Define a fixed target chip.

Parameters

- **name** (*str*) – Description of the chip.
- **num_steps** (*List[int] | Tuple[int]*) – Number of windows in each block.
- **step_size** (*List[float] | Tuple[float]*) – Size of each window (distance between the centers in x and y direction).
- **num_blocks** (*List[int] | Tuple[int]*) – Total number of blocks in the chip.
- **block_size** (*List[int] | Tuple[int]*) – Size of each block.
- **start_pos** (*List[float]*) – Start coordinates (x,y,z)

nexgen.beamlines.SSX_chip.**compute_goniometer**(*chip: Chip, blocks: Dict | None = None, full: bool = False, ax1: str = 'sam_y', ax2: str = 'sam_x'*) → Dict[Dict[str | Tuple, float | int]]

Compute the start coordinates of each block in a chip scan.

The function returns a dictionary associating a list of axes start values and a scan direction to each scanned block. If full is True, the blocks argument will be overridden and coordinates will be calculated for every block in the chip.

Parameters

- **chip** (Chip) – General description of the chip schematics: number and size of blocks, size and step of each window, start positions.
- **blocks** (*Dict | None, optional*) – Scanned blocks. Defaults to None.
- **full** (*bool, optional*) – True if all blocks have been scanned. Defaults to False.
- **ax1** (*str, optional*) – Axis name corrsponding to slow varying axis. Defaults to "sam_y".
- **ax2** (*str, optional*) – Axis name corrsponding to fast varying axis. Defaults to "sam_x".

Returns

Axes start coordinates and scan direction of each block. eg. {

'01'/(0,0): {
'ax1': 0.0, 'ax2': 0.0, 'direction': 1,

}

}

Return type
Dict[Dict[str | Tuple, float | int]]

nexgen.beamlines.SSX_chip.**fullchip_conversion_table**(*chip: Chip*) → Dict

Associate block coordinates to block number for a full chip.

Parameters
chip (Chip) – General description of the chip.

Returns
Conversion table, keys are block numbers, values are coordinates.

Return type
Dict

nexgen.beamlines.SSX_chip.**read_chip_map**(*mapfile: Path | str*, *x_blocks: int*, *y_blocks: int*) → Dict

Read the .map file for the current collection on a chip.

Parameters

- **mapfile** (*Path | str*) – Path to .map file. If None, assumes fullchip.
- **x_blocks** (*int*) – Total number of blocks in x direction in the chip.
- **y_blocks** (*int*) – Total number of blocks in y direction in the chip.

Returns
A dictionary whose values indicate either the coordinates on the chip of the scanned blocks, or a string indicating that the whole chip is being scanned.

Return type
Dict

4.6 Serial crystallography: experiment types

nexgen.beamlines.SSX_expt.**run_extruder**(*goniometer_axes: List[Axis]*, *num_imgs: int*, *pump_probe: PumpProbe*, *osc_axis: str = 'omega'*) → Tuple[List, Dict, Dict]

Run the goniometer computations for an extruder experiment.

Parameters

- **goniometer_axes** (*List[Axis]*) – List of goniometer axes for current beamline.
- **num_imgs** (*int*) – Total number of images.
- **pump_probe** (PumpProbe) – Pump probe parameters.
- **osc_axis** – Defines which axis is considered the "moving" one. Defaults to omega.

Returns
goniometer_axes: updated goniometer_axes list with actual values from the scan. SCAN: dictionary with oscillation scan axis values. pump_info: updated pump probe information.

Return type
Tuple[List, Dict, Dict]

nexgen.beamlines.SSX_expt.**run_fixed_target**(*goniometer_axes: List[Axis]*, *chip_info: Dict[str, List]*, *chipmap: Path | str*, *pump_probe:* PumpProbe, *scan_axes: List[str, str] = ['sam_y', 'sam_x']*) → Tuple[Dict, Dict]

Run the goniometer computations for a fixed-target experiment.

Parameters

- **goniometer_axes** (*List[Axis]*) – List of goniometer axes for current beamline.
- **chip_info** (*Dict[str, List]*) – General information about the chip: number and size of blocks, size and step of each window, start positions, number of exposures.
- **chipmap** (*Path | str*) – Path to .map file. If None is passed, assumes a fullchip.
- **pump_probe** (PumpProbe) – Pump probe parameters.
- **scan_axes** (*List[str, str], optional*) – List of scan axes, in order slow,fast. Defaults to [“sam_y”, “sam_x”].

Raises

- **ValueError** – If one or both of the axes names passed as input are not part of the goniometer axes.
- **ValueError** – if chip_info hasn’t been passed or is an empty dictionary.

Returns
SCAN: Dictionary with grid scan values. pump_info: Updated pump probe information.

Return type
Tuple[Dict, Dict]

4.7 Electron diffraction: Singla writer

nexgen.beamlines.ED_singla_nxs.**singla_nexus_writer**(*master_file: ~pathlib.Path | str*, *det_distance: float*, *exp_time: float*, *ED_cs: ~typing.Dict = {'convention': 'ED', 'origin': (0, 0, 0), 'x': Axis(name='x', depends='.', transformation_type=<TransformationType.TRANSLATION: 'translation'>, vector=(0, 1, 0), start_pos=0.0, increment=0.0, num_steps=0, offset=(0.0, 0.0, 0.0)), 'y': Axis(name='y', depends='x', transformation_type=<TransformationType.TRANSLATION: 'translation'>, vector=(-1, 0, 0), start_pos=0.0, increment=0.0, num_steps=0, offset=(0.0, 0.0, 0.0)), 'z': Axis(name='z', depends='y', transformation_type=<TransformationType.TRANSLATION: 'translation'>, vector=(0, 0, 1), start_pos=0.0, increment=0.0, num_steps=0, offset=(0.0, 0.0, 0.0))}*, *datafiles: ~typing.List[~pathlib.Path | str] | None = None*, *convert2mcstas: bool = False*, ***params*)

Gather all collection parameters and write the NeXus file for an electron diffraction collectio using SINGLA detector.

Parameters

- **master_file** (*Path | str*) – Singla master file.
- **det_distance** (*float*) – Sample-detector distance, in mm.
- **exp_time** (*float*) – Exposure time, in s.
- **ED_cs** (*Dict, optional*) – Definition of the ED coordinate system in use. Defaults to {"convention": "ED", "origin": (0, 0, 0), "x": Axis("x", ".", "translation", [0, 1, 0]), "y": Axis("y", "x", "translation", [-1, 0, 0]), "z": Axis("z", "y", "translation", [0, 0, 1]),}
- **datafiles** (*List[Path | str], optional*) – List of data files. Defaults to None.
- **convert2mcstas** (*bool, optional*) – Convert vectors to mcstas if required. Defaults to False.

Keyword Arguments

- **n_imgs** (*int*) – Total number of images in collection.
- **scan_axis** (*List[str, float, float]*) – Rotation axis name, start and increment.
- **outdir** (*Path | str*) – Directory where to save the file. Only specify if different from meta_file directory.
- **beam_center** (*List[float, float]*) – Beam center position, in pixels.
- **wavelength** (*float*) – Wavelength of incident beam, in A.
- **start_time** (*datetime*) – Experiment start time.
- **new_source_info** (*Dict*) – Information about Source that might differ from the default. eg. {"facility_id": "MICROSCOPE", "name": "Not Diamond"}
- **vds_writer** (*str*) – Write dataset or external file.

4.8 GDA integration tools

4.8.1 Read geometry and detector parameters from GDA-generated JSON files

Tools to extract goniometer and detector parameters from GDA JSON files.

class nexgen.beamlines.GDAtools.GDAjson2params.**JSONParamsIO**(*json_file: Path | str*)

Read JSON file and exctract parameters.

get_coordinate_frame() → str

Get the coordinate frame from geometry json file.

get_detector_axes_from_file() → List[*Axis*]

Read the detector axes information from the GDA-supplied json file.

get_detector_params_from_file() → *EigerDetector* | *TristanDetector* | *SinglaDetector* | *JungfrauDetector* | *CetaDetector*

Read the detector parameters from the GDA-supplied json file.

get_fast_and_slow_direction_vectors_from_file(*det_type: str*) → Tuple[*Point3D*, *Point3D*]

Read detector fast and slow axes from the GDA-supplied json file.

get_goniometer_axes_from_file() → List[*Axis*]

Read the axes information from the GDA-supplied json file.

4.8.2 Gather beamline and collection information from GDA-generated xml file

IO tool to gather beamline and collection information from xml file.

class nexgen.beamlines.GDAtools.ExtendedRequest.**ExtendedRequestIO**(*xmlfile: Path | str*)

Define an ExtendedRequest object which in GDA gathers all the information regarding beamline and collection into an xml file.

nexgen.beamlines.GDAtools.ExtendedRequest.**read_det_position_from_xml**(*ecr:* ExtendedRequestIO, *det_description: str*) → List[float]

Extract the detector position contained in the xml file.

Parameters

- **ecr** (ExtendedRequestIO) – XML tree parser.
- **det_description** (*str*) – Detector description

Returns
Detector axes positions in the order [2theta, det_z]

Return type
List[float]

nexgen.beamlines.GDAtools.ExtendedRequest.**read_scan_from_xml**(*ecr:* ExtendedRequestIO)

Extract information about the scan contained in the xml file.

Parameters

- **ecr** (ExtendedRequestIO) – XML tree parser.
- **xmlfile** (*Path | str*) – Path to xml file.

Returns
Name of the rotation scan axis pos (Dict): Dictionary containing the (start,end,increment) values for each goniometer axis. num (int): Number of images written.

Return type
scan_axis (str)

PYTHON MODULE INDEX

n

INDEX

E

F

G

H

I

U

V

W

X