

---

# **nexgen Documentation**

***Release 0.9.1***

**Diamond Light Source - Scientific Software**

**Apr 22, 2024**



## CONTENTS:

<b>1</b>	<b>Usage</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Command line tools . . . . .	3
<b>2</b>	<b>Beamline specific utilities</b>	<b>7</b>
2.1	Time resolved collections on I19-2 . . . . .	7
2.2	Serial crystallography . . . . .	9
2.3	I19-2 CLI . . . . .	10
2.4	SSX CLI . . . . .	11
<b>3</b>	<b>API</b>	<b>13</b>
3.1	Defining the various parts of a nexus file . . . . .	13
3.2	Writing tools . . . . .	23
3.3	Copying tools . . . . .	31
3.4	Utilities . . . . .	34
3.5	Logging configuration . . . . .	41
<b>4</b>	<b>Beamlines API</b>	<b>43</b>
4.1	General utilities . . . . .	43
4.2	I19-2 . . . . .	43
4.3	Serial crystallography: Eiger writers . . . . .	49
4.4	Serial crystallography: Tristan writers . . . . .	50
4.5	Serial crystallography: chip tools . . . . .	51
4.6	Serial crystallography: experiment types . . . . .	52
4.7	Electron diffraction: Singla writer . . . . .	53
4.8	GDA integration tools . . . . .	54
	<b>Python Module Index</b>	<b>57</b>
	<b>Index</b>	<b>59</b>



- 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](#).



## 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. The 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 the 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 one 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 original 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 for 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
```

## 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 command 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
```





## 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 collection. 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 default 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 =  
'I')
```

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/\\_diffn\\_source\\_type.html](https://mmcif.wwpdb.org/dictionaries/mmcif_pdbx_v50.dic/items/_diffn_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')
```

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”.

**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.nxm_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'*)

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

This method overrides the write() method of NXmxFileWriter, from which this 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”.

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.nxm_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 this 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 this 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(nxfile: 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

- **nxfile** (*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 `mctas`.
- **data\_type** (*Tuple[str, int]*) – Images or event-mode data, and eventually how many are being written.
- (**Dict[str, Any]** *(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 ...

```
nextgen.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)

```
nextgen.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, defines 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 nextgen.tools.vds_tools.Dataset(name: 'str', source_shape: 'Tuple[int]', start_index: 'int' = 0,
                                     stop_index: 'int' = 1000, dest_shape: 'Tuple[int]' = None)
```

```
nextgen.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.

```
nextgen.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*)

```
nextgen.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*)

```
nextgen.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.

```
nextgen.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.

```
nextgen.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 length.
- **ValueError** – If 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 (`.nxh`) 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 length.

**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 length.

**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.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 associated 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 the `{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 corner 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. 'NX-data' 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**

**nx\_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.



## 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 interpreter/ 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 Å.
- **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 only 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 to 0.0. Only really needed for Tristan collections.

**id**

Axis name.

**inc**

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

**start**

Axis start position. Defaults to 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 Å.

- **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.\n  ↪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 Å.

**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 Å.
- **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 collection 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 Å.

- **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 corresponding to slow varying axis. Defaults to "sam\_y".
- **ax2** (*str, optional*) – Axis name corresponding 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 Å.
- **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 extract 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

- `nexgen.beamlines.GDATools.ExtendedRequest`, 55
- `nexgen.beamlines.GDATools.GDAjson2params`, 54
- `nexgen.beamlines.SSX_chip`, 51
- `nexgen.log`, 41
- `nexgen.nxs_copy.copy_nexus`, 31
- `nexgen.nxs_copy.copy_tristan_nexus`, 32
- `nexgen.nxs_copy.copy_utils`, 37
- `nexgen.nxs_utils.axes`, 13
- `nexgen.nxs_utils.detector`, 16
- `nexgen.nxs_utils.goniometer`, 16
- `nexgen.nxs_utils.sample`, 23
- `nexgen.nxs_utils.scan_utils`, 14
- `nexgen.nxs_utils.source`, 22
- `nexgen.nxs_write.write_utils`, 35
- `nexgen.tools.metafile`, 40
- `nexgen.tools.vds_tools`, 29
- `nexgen.utils`, 34



## A

`add_NXnote()` (*nexgen.nxs\_write.nxm\_writer.EDNXmxFileWriter* *gen.nxs\_copy.copy\_utils*), 37  
method), 25

`add_NXnote()` (*nexgen.nxs\_write.nxm\_writer.NXmxFileWriter* *gen.nxs\_copy.copy\_utils*), 38  
method), 23

`Attenuator` (class in *nexgen.nxs\_utils.source*), 22

`axes` (class in *nexgen.beamlines.I19\_2\_nxs*), 45

`axes_list` (*nexgen.nxs\_utils.goniometer.Goniometer* attribute), 16

`axes_order` (*nexgen.nxs\_utils.scan\_utils.GridScanOptions* attribute), 16

`Axis` (class in *nexgen.nxs\_utils.axes*), 13

## B

`Beam` (class in *nexgen.nxs\_utils.source*), 22

`beam_center` (*nexgen.beamlines.I19\_2\_gda\_nxs.tr\_collect* attribute), 47

`beam_center` (*nexgen.beamlines.I19\_2\_nxs.CollectionParams* attribute), 47

`beam_center` (*nexgen.nxs\_utils.detector.Detector* attribute), 17

`BeamlineAxes` (class in *nexgen.beamlines.beamline\_utils*), 43

`build_a_tristan()` (in module *nexgen.tools.data\_writer*), 27

`build_an_eiger()` (in module *nexgen.tools.data\_writer*), 27

## C

`calculate_origin()` (in module *nexgen.nxs\_write.write\_utils*), 35

`calculate_scan_points()` (in module *nexgen.nxs\_utils.scan\_utils*), 14

`call_writers()` (in module *nexgen.command\_line.cli\_utils*), 26

`CetaDetector` (class in *nexgen.nxs\_utils.detector*), 16

`Chip` (class in *nexgen.beamlines.SSX\_chip*), 51

`clean_unused_links()` (in module *nexgen.tools.vds\_tools*), 29

`compute_goniometer()` (in module *nexgen.beamlines.SSX\_chip*), 51

`compute_ssx_axes()` (in module *nexgen.nxs\_copy.copy\_utils*), 37

`config()` (in module *nexgen.log*), 41

`convert_scan_axis()` (in module *nexgen.nxs\_copy.copy\_utils*), 38

`create_attributes()` (in module *nexgen.nxs\_write.write\_utils*), 36

`create_virtual_layout()` (in module *nexgen.tools.vds\_tools*), 29

## D

`Dataset` (class in *nexgen.tools.vds\_tools*), 29

`DectrisMetafile` (class in *nexgen.tools.metafile*), 40

`define_scan_axes_for_event_mode()` (*nexgen.nxs\_utils.goniometer.Goniometer* method), 16

`define_scan_from_goniometer_axes()` (*nexgen.nxs\_utils.goniometer.Goniometer* method), 16

`define_vds_data_type()` (in module *nexgen.tools.meta\_reader*), 40

`depends` (*nexgen.nxs\_utils.axes.Axis* attribute), 13

`description` (*nexgen.nxs\_utils.detector.EigerDetector* attribute), 17

`description` (*nexgen.nxs\_utils.detector.JungfrauDetector* attribute), 18

`description` (*nexgen.nxs\_utils.detector.SinglaDetector* attribute), 20

`description` (*nexgen.nxs\_utils.detector.TristanDetector* attribute), 21

`det_axes` (class in *nexgen.beamlines.I19\_2\_nxs*), 45

`Detector` (class in *nexgen.nxs\_utils.detector*), 16

`detector_axes` (*nexgen.nxs\_utils.detector.Detector* attribute), 17

`detector_json` (*nexgen.beamlines.I19\_2\_gda\_nxs.tr\_collect* attribute), 47

`detector_name` (*nexgen.beamlines.I19\_2\_gda\_nxs.tr\_collect* attribute), 48

`detector_name` (*nexgen.beamlines.I19\_2\_nxs.CollectionParams* attribute), 47

`detector_params` (*nexgen.nxs\_utils.detector.Detector* attribute), 17

[detector\\_type \(nexgen.nxs\\_utils.detector.EigerDetector attribute\), 18](#)  
[detector\\_type \(nexgen.nxs\\_utils.detector.JungfrauDetector attribute\), 19](#)  
[detector\\_type \(nexgen.nxs\\_utils.detector.SinglaDetector attribute\), 20](#)  
[detector\\_type \(nexgen.nxs\\_utils.detector.TristanDetector attribute\), 21](#)  
[DetectorModule \(class in nexgen.nxs\\_utils.detector\), 17](#)  
**E**  
[EDNXmxFileWriter \(class in nexgen.nxs\\_write.nxm\\_x\\_writer\), 24](#)  
[eiger\\_writer\(\) \(in module nexgen.beamlines.I19\\_2\\_gda\\_nxs\), 48](#)  
[eiger\\_writer\(\) \(in module nexgen.beamlines.I19\\_2\\_nxs\), 44](#)  
[EigerDetector \(class in nexgen.nxs\\_utils.detector\), 17](#)  
[end \(nexgen.beamlines.I19\\_2\\_nxs.axes attribute\), 45](#)  
[EventNXmxFileWriter \(class in nexgen.nxs\\_write.nxm\\_x\\_writer\), 24](#)  
[exp\\_time \(nexgen.nxs\\_utils.detector.Detector attribute\), 17](#)  
[exposure\\_time \(nexgen.beamlines.I19\\_2\\_gda\\_nxs.tr\\_collect attribute\), 48](#)  
[exposure\\_time \(nexgen.beamlines.I19\\_2\\_nxs.CollectionParams attribute\), 47](#)  
[ExtendedRequestIO \(class in nexgen.beamlines.GDAtools.ExtendedRequest\), 55](#)  
**F**  
[Facility \(class in nexgen.nxs\\_utils.source\), 22](#)  
[fast\\_axis \(nexgen.nxs\\_utils.detector.DetectorModule attribute\), 17](#)  
[find\\_datasets\\_in\\_file\(\) \(in module nexgen.tools.vds\\_tools\), 29](#)  
[find\\_number\\_of\\_images\(\) \(in module nexgen.nxs\\_write.write\\_utils\), 36](#)  
[fullchip\\_conversion\\_table\(\) \(in module nexgen.beamlines.SSX\\_chip\), 52](#)  
**G**  
[generate\\_event\\_files\(\) \(in module nexgen.tools.data\\_writer\), 28](#)  
[generate\\_image\\_files\(\) \(in module nexgen.tools.data\\_writer\), 28](#)  
[geometry\\_json \(nexgen.beamlines.I19\\_2\\_gda\\_nxs.tr\\_collect attribute\), 48](#)  
[get\\_coordinate\\_frame\(\) \(nexgen.beamlines.GDAtools.GDAjson2params.JSONParamsIO attribute\), 18](#)  
[get\\_detector\\_axes\\_from\\_file\(\) \(nexgen.beamlines.GDAtools.GDAjson2params.JSONParamsIO method\), 54](#)  
[get\\_detector\\_description\(\) \(nexgen.nxs\\_utils.detector.Detector method\), 17](#)  
[get\\_detector\\_mode\(\) \(nexgen.nxs\\_utils.detector.Detector method\), 17](#)  
[get\\_detector\\_params\\_from\\_file\(\) \(nexgen.beamlines.GDAtools.GDAjson2params.JSONParamsIO method\), 54](#)  
[get\\_fast\\_and\\_slow\\_direction\\_vectors\\_from\\_file\(\) \(nexgen.beamlines.GDAtools.GDAjson2params.JSONParamsIO method\), 54](#)  
[get\\_filename\\_template\(\) \(in module nexgen.utils\), 34](#)  
[get\\_goniometer\\_axes\\_from\\_file\(\) \(nexgen.beamlines.GDAtools.GDAjson2params.JSONParamsIO method\), 54](#)  
[get\\_iso\\_timestamp\(\) \(in module nexgen.utils\), 34](#)  
[get\\_module\\_info\(\) \(nexgen.nxs\\_utils.detector.Detector method\), 17](#)  
[get\\_nexus\\_filename\(\) \(in module nexgen.utils\), 34](#)  
[get\\_nexus\\_tree\(\) \(in module nexgen.nxs\\_copy.copy\\_utils\), 38](#)  
[get\\_number\\_of\\_scan\\_points\(\) \(nexgen.nxs\\_utils.goniometer.Goniometer method\), 16](#)  
[get\\_skip\\_list\(\) \(in module nexgen.nxs\\_copy.copy\\_utils\), 38](#)  
[Goniometer \(class in nexgen.nxs\\_utils.goniometer\), 16](#)  
[GridScanOptions \(class in nexgen.nxs\\_utils.scan\\_utils\), 15](#)  
**H**  
[h5str\(\) \(in module nexgen.nxs\\_copy.copy\\_utils\), 39](#)  
**I**  
[id \(nexgen.beamlines.I19\\_2\\_nxs.axes attribute\), 45](#)  
[id \(nexgen.beamlines.I19\\_2\\_nxs.det\\_axes attribute\), 45](#)  
[id \(nexgen.nxs\\_utils.source.Facility attribute\), 22](#)  
[identify\\_grid\\_scan\\_axes\(\) \(in module nexgen.nxs\\_utils.scan\\_utils\), 15](#)  
[identify\\_osc\\_axis\(\) \(in module nexgen.nxs\\_utils.scan\\_utils\), 15](#)  
[identify\\_tristan\\_scan\\_axis\(\) \(in module nexgen.nxs\\_copy.copy\\_utils\), 39](#)  
[image\\_size \(nexgen.nxs\\_utils.detector.EigerDetector attribute\), 18](#)  
[image\\_size \(nexgen.nxs\\_utils.detector.JungfrauDetector attribute\), 19](#)



`image_size` (*nexgen.nxs\_utils.detector.SinglaDetector attribute*), 20  
`image_size` (*nexgen.nxs\_utils.detector.TristanDetector attribute*), 21  
`image_vds_writer()` (in module *nexgen.tools.vds\_tools*), 29  
`images_nexus()` (in module *nexgen.nxs\_copy.copy\_nexus*), 31  
`inc` (*nexgen.beamlines.I19\_2\_nxs.axes attribute*), 45  
`increment` (*nexgen.nxs\_utils.axes.Axis attribute*), 14  
`is_chipmap_in_tristan_nxs()` (in module *nexgen.nxs\_copy.copy\_utils*), 39

## J

`JSONParamsIO` (class in *nexgen.beamlines.GDAtools.GDAjson2params*), 54  
`jungfrau_vds_writer()` (in module *nexgen.tools.vds\_tools*), 30  
`JungfrauDetector` (class in *nexgen.nxs\_utils.detector*), 18

## L

`LoggingContext` (class in *nexgen.log*), 41

## M

`mask_and_flatfield_writer()` (in module *nexgen.nxs\_write.write\_utils*), 36  
`meta_file` (*nexgen.beamlines.I19\_2\_gda\_nxs.tr\_collect attribute*), 48  
`metafile` (*nexgen.beamlines.I19\_2\_nxs.CollectionParams attribute*), 47  
`mode` (*nexgen.nxs\_utils.detector.TristanDetector attribute*), 21  
`module`  
    *nexgen.beamlines.GDAtools.ExtendedRequest*, 55  
    *nexgen.beamlines.GDAtools.GDAjson2params*, 54  
    *nexgen.beamlines.SSX\_chip*, 51  
    *nexgen.log*, 41  
    *nexgen.nxs\_copy.copy\_nexus*, 31  
    *nexgen.nxs\_copy.copy\_tristan\_nexus*, 32  
    *nexgen.nxs\_copy.copy\_utils*, 37  
    *nexgen.nxs\_utils.axes*, 13  
    *nexgen.nxs\_utils.detector*, 16  
    *nexgen.nxs\_utils.goniometer*, 16  
    *nexgen.nxs\_utils.sample*, 23  
    *nexgen.nxs\_utils.scan\_utils*, 14  
    *nexgen.nxs\_utils.source*, 22  
    *nexgen.nxs\_write.write\_utils*, 35  
    *nexgen.tools.metafile*, 40  
    *nexgen.tools.vds\_tools*, 29  
    *nexgen.utils*, 34

`module` (*nexgen.nxs\_utils.detector.Detector attribute*), 17  
`multiple_images_nexus()` (in module *nexgen.nxs\_copy.copy\_tristan\_nexus*), 32

## N

`name` (*nexgen.nxs\_utils.axes.Axis attribute*), 13  
`name` (*nexgen.nxs\_utils.source.Facility attribute*), 22  
*nexgen.beamlines.GDAtools.ExtendedRequest module*, 55  
*nexgen.beamlines.GDAtools.GDAjson2params module*, 54  
*nexgen.beamlines.SSX\_chip module*, 51  
*nexgen.log module*, 41  
*nexgen.nxs\_copy.copy\_nexus module*, 31  
*nexgen.nxs\_copy.copy\_tristan\_nexus module*, 32  
*nexgen.nxs\_copy.copy\_utils module*, 37  
*nexgen.nxs\_utils.axes module*, 13  
*nexgen.nxs\_utils.detector module*, 16  
*nexgen.nxs\_utils.goniometer module*, 16  
*nexgen.nxs\_utils.sample module*, 23  
*nexgen.nxs\_utils.scan\_utils module*, 14  
*nexgen.nxs\_utils.source module*, 22  
*nexgen.nxs\_write.write\_utils module*, 35  
*nexgen.tools.metafile module*, 40  
*nexgen.tools.vds\_tools module*, 29  
*nexgen.utils module*, 34  
`nexus_writer()` (in module *nexgen.beamlines.I19\_2\_nxs*), 43  
`num_steps` (*nexgen.nxs\_utils.axes.Axis attribute*), 14  
`NXmxFileWriter` (class in *nexgen.nxs\_write.nxm\_writer*), 23

## O

`offset` (*nexgen.nxs\_utils.axes.Axis attribute*), 14  
`overload` (*nexgen.nxs\_utils.detector.EigerDetector attribute*), 18  
`overload` (*nexgen.nxs\_utils.detector.JungfrauDetector attribute*), 19

`overload` (*nexgen.nxs\_utils.detector.SinglaDetector* attribute), 20

## P

`pixel_size` (*nexgen.nxs\_utils.detector.EigerDetector* attribute), 18

`pixel_size` (*nexgen.nxs\_utils.detector.JungfrauDetector* attribute), 19

`pixel_size` (*nexgen.nxs\_utils.detector.SinglaDetector* attribute), 20

`pixel_size` (*nexgen.nxs\_utils.detector.TristanDetector* attribute), 21

`Point3D` (class in *nexgen.utils*), 13

`pseudo_event_list()` (in module *nexgen.tools.data\_writer*), 28

`pseudo_events_nexus()` (in module *nexgen.nxs\_copy.copy\_nexus*), 32

`PumpProbe` (class in *nexgen.beamlines.beamline\_utils*), 43

## R

`read_chip_map()` (in module *nexgen.beamlines.SSX\_chip*), 52

`read_det_position_from_xml()` (in module *nexgen.beamlines.GDAtools.ExtendedRequest*), 55

`read_scan_from_xml()` (in module *nexgen.beamlines.GDAtools.ExtendedRequest*), 55

`run_extruder()` (in module *nexgen.beamlines.SSX\_expt*), 52

`run_fixed_target()` (in module *nexgen.beamlines.SSX\_expt*), 53

## S

`Sample` (class in *nexgen.nxs\_utils.sample*), 23

`scan` (*nexgen.nxs\_utils.goniometer.Goniometer* attribute), 16

`scan_axis` (*nexgen.beamlines.I19\_2\_nxs.CollectionParams* attribute), 47

`ScanAxisError`, 16

`ScanAxisNotFoundError`, 16

`sensor_material` (*nexgen.nxs\_utils.detector.EigerDetector* attribute), 18

`sensor_material` (*nexgen.nxs\_utils.detector.JungfrauDetector* attribute), 19

`sensor_material` (*nexgen.nxs\_utils.detector.SinglaDetector* attribute), 20

`sensor_material` (*nexgen.nxs\_utils.detector.TristanDetector* attribute), 21

`sensor_thickness` (*nexgen.nxs\_utils.detector.JungfrauDetector* attribute), 19

`sensor_thickness` (*nexgen.nxs\_utils.detector.SinglaDetector* attribute), 20

`sensor_thickness` (*nexgen.nxs\_utils.detector.TristanDetector* attribute), 21

`serial_images_nexus()` (in module *nexgen.nxs\_copy.copy\_tristan\_nexus*), 33

`set_dependency()` (in module *nexgen.nxs\_write.write\_utils*), 36

`set_instrument_name()` (*nexgen.nxs\_utils.source.Source* method), 22

`short_name` (*nexgen.nxs\_utils.source.Facility* attribute), 22

`singla_nexus_writer()` (in module *nexgen.beamlines.ED\_singla\_nxs*), 53

`SinglaDetector` (class in *nexgen.nxs\_utils.detector*), 19

`single_image_nexus()` (in module *nexgen.nxs\_copy.copy\_tristan\_nexus*), 33

`slow_axis` (*nexgen.nxs\_utils.detector.DetectorModule* attribute), 17

`snaked` (*nexgen.nxs\_utils.scan\_utils.GridScanOptions* attribute), 16

`Source` (class in *nexgen.nxs\_utils.source*), 22

`split_datasets()` (in module *nexgen.tools.vds\_tools*), 30

`ssx_eiger_writer()` (in module *nexgen.beamlines.SSX\_Eiger\_nxs*), 49

`ssx_tristan_writer()` (in module *nexgen.beamlines.SSX\_Tristan\_nxs*), 50

`start` (*nexgen.beamlines.I19\_2\_nxs.axes* attribute), 45

`start` (*nexgen.beamlines.I19\_2\_nxs.det\_axes* attribute), 45

`start_pos` (*nexgen.nxs\_utils.axes.Axis* attribute), 13

`start_time` (*nexgen.beamlines.I19\_2\_gda\_nxs.tr\_collect* attribute), 48

`stop_time` (*nexgen.beamlines.I19\_2\_gda\_nxs.tr\_collect* attribute), 48

## T

`to_dict()` (*nexgen.nxs\_utils.source.Source* method), 22

`tot_num_images` (*nexgen.beamlines.I19\_2\_nxs.CollectionParams* attribute), 47

`tr_collect` (class in *nexgen.beamlines.I19\_2\_gda\_nxs*), 47

`transformation_type` (*nexgen.nxs\_utils.axes.Axis* attribute), 13

`TransformationType` (class in *nexgen.nxs\_utils.axes*), 14

`transmission(nexgen.beamlines.I19_2_nxs.CollectionParameters, attribute), 47`  
`tristan_writer()` (in module `nexgen.beamlines.I19_2_gda_nxs`), 48  
`tristan_writer()` (in module `nexgen.beamlines.I19_2_nxs`), 44  
`TristanDetector` (class in `nexgen.nxs_utils.detector`), 20  
`TristanMetafile` (class in `nexgen.tools.metafile`), 40  
`type` (`nexgen.nxs_utils.source.Facility` attribute), 22

## U

`underload` (`nexgen.nxs_utils.detector.EigerDetector` attribute), 18  
`underload` (`nexgen.nxs_utils.detector.JungfrauDetector` attribute), 19  
`underload` (`nexgen.nxs_utils.detector.SinglaDetector` attribute), 20  
`units_of_length()` (in module `nexgen.utils`), 34  
`units_of_time()` (in module `nexgen.utils`), 35  
`UnknownDetectorTypeError`, 21  
`update_axes_from_meta()` (in module `nexgen.tools.meta_reader`), 40  
`update_timestamps()` (`nexgen.nxs_write.nxm_writer.EDNXmxFileWriter` method), 25  
`update_timestamps()` (`nexgen.nxs_write.nxm_writer.NXmxFileWriter` method), 23

## V

`vds_file_writer()` (in module `nexgen.tools.vds_tools`), 31  
`vector` (`nexgen.nxs_utils.axes.Axis` attribute), 13

## W

`walk_nxs()` (in module `nexgen.utils`), 35  
`wavelength` (`nexgen.beamlines.I19_2_gda_nxs.tr_collect` attribute), 48  
`wavelength` (`nexgen.beamlines.I19_2_nxs.CollectionParameters` attribute), 47  
`with_traceback()` (`nexgen.nxs_utils.detector.UnknownDetectorTypeError` method), 21  
`write()` (`nexgen.nxs_write.nxm_writer.EDNXmxFileWriter` method), 25  
`write()` (`nexgen.nxs_write.nxm_writer.EventNXmxFileWriter` method), 24  
`write()` (`nexgen.nxs_write.nxm_writer.NXmxFileWriter` method), 23  
`write_compressed_copy()` (in module `nexgen.nxs_write.write_utils`), 37  
`write_vds()` (`nexgen.nxs_write.nxm_writer.EDNXmxFileWriter` method), 25