



---

# The CXI File Format for Coherent X-ray Imaging

---

Version 1.3

Filipe R. N. C. Maia  
January 10, 2012

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Goals . . . . .	1
<b>2</b>	<b>The Design of the CXI</b>	<b>1</b>
2.1	HDF5 . . . . .	1
2.2	NeXus . . . . .	2
2.3	CXI . . . . .	2
<b>3</b>	<b>CXI by example</b>	<b>3</b>
3.1	A minimal CXI file . . . . .	3
3.2	A typical raw data CXI file . . . . .	4
<b>4</b>	<b>NeXus compatibility</b>	<b>5</b>
<b>5</b>	<b>Datatypes</b>	<b>6</b>
5.1	Complex Numbers . . . . .	6
<b>6</b>	<b>Default units for CXI entries</b>	<b>7</b>
6.1	Angles . . . . .	7
6.2	Dates . . . . .	7
<b>7</b>	<b>Memory Layout</b>	<b>8</b>
<b>8</b>	<b>Geometry</b>	<b>8</b>
8.1	Coordinate System . . . . .	8
8.2	The local coordinate system of objects . . . . .	8
8.2.1	The orientation of pixel detectors . . . . .	9
<b>9</b>	<b>Scans</b>	<b>10</b>
9.1	Dimension Scales . . . . .	11
<b>A</b>	<b>More CXI file examples</b>	<b>12</b>
A.1	CXI file from a ptychography experiment . . . . .	12
A.2	CXI file after initial analysis of a diffraction image . . . . .	13
A.3	CXI file of a phased 3D image . . . . .	14
<b>B</b>	<b>CXI entries reference</b>	<b>15</b>
B.1	Top level (root) . . . . .	15
B.2	Attenuator . . . . .	15
B.3	Data . . . . .	16

B.4	Detector . . . . .	16
B.5	Entry . . . . .	18
B.6	Geometry . . . . .	19
B.7	Image . . . . .	20
B.8	Instrument . . . . .	22
B.9	Monochromator . . . . .	23
B.10	Note . . . . .	23
B.11	Process . . . . .	24
B.12	Sample . . . . .	24
B.13	Source . . . . .	25
<b>C</b>	<b>Changes from previous versions</b>	<b>27</b>
C.1	Version 1.3 . . . . .	27
C.2	Version 1.2.1 . . . . .	27
C.3	Version 1.2 . . . . .	27
C.4	Version 1.1 . . . . .	27
C.5	Version 1.0 . . . . .	28
<b>D</b>	<b>Diagrams color code</b>	<b>29</b>
<b>E</b>	<b>Code examples</b>	<b>30</b>
E.1	Creating a minimal CXI file . . . . .	30
E.2	Creating a typical raw CXI file . . . . .	31
E.3	NeXus compatible version of the typical raw CXI file . . . .	33
E.4	3D complex valued Image file . . . . .	35
E.5	Spartan viewer for CXI files . . . . .	37

## 1 Introduction

The CXI file format was created as common format for all the data in the Coherent X-ray Imaging Data Bank (CXIDB). Naturally its scope is all experimental data collected during Coherent X-ray Imaging experiments as well as all data generated during the analysis of the experimental data.

### 1.1 Goals

The CXI file format aims to create a data format with the following requirements:

1. Simple - both writing and reading should be made simple.
2. Flexible - users should be able to easily extend it.
3. Fast - it should be efficient so as not to become a bottleneck.
4. Extendable - new features should be easily added without breaking compatibility with previous versions.
5. Unambiguous - it should be possible to interpret the files without using external information.
6. Compatible - the format should be as compatible as possible with existing formats

These are hard and often contradicting requirements (e.g. simple and unambiguous). When such conflicts occur the highest ranked goal is often preferred (e.g. simple).

## 2 The Design of the CXI

### 2.1 HDF5

The HDF5 format is the basis of CXI format. CXI is not really a completely new file format but simply a set of rules designed to create HDF5 files with a common structure and to allow a uniform and consistent interpretation of such files.

HDF5 was chosen as the basis because it is a widely used high performance scientific data format which many programs can already, at

least partially, read and write. It also brings with it the almost automatic fulfilment of requirements 2, 3 and 4. HDF5 version 1.8 or higher is required as previous versions don't support all features required by CXI.

## 2.2 NeXus

Another important influence in the design of CXI is the NeXus file format for neutron, x-ray and muon science. As CXI, NeXus is also based on HDF5 (although it can use others basis formats such as XML) and shares many of the goals of CXI, with one important exception - it is *not* a simple format. All NeXus file require attributes in HDF5 files and using many existing programs it is laborious and hard, if not outright impossible, to create such attributes. The large scope of NeXus also results in files that can be complex to read and interpret.

NeXus classes are the fundamental pieces that make up NeXus files. The documentation for the classes can be found in <http://download.nexusformat.org/doc/html/ClassDefinitions.html>. Each class has a set of members which can either be data fields with well defined members (e.g. the `data` field of the `NXdata` class), or they can be other classes in which case the name is arbitrary. Each NeXus class corresponds to an HDF5 group with one or more attributes (class name, units, long name, etc...) and each class data field corresponds to an HDF5 dataset.

## 2.3 CXI

The approach used in CXI is to use HDF5 as basis format and adopt a small set of rules derived from the NeXus format with some additional restraints which should make the format simpler to write and interpret. This allows us to relax the format, making it simple for new users, while still retaining enough information to reconstruct an NeXus compile file if necessary.

Below are the main rules used in CXI files:

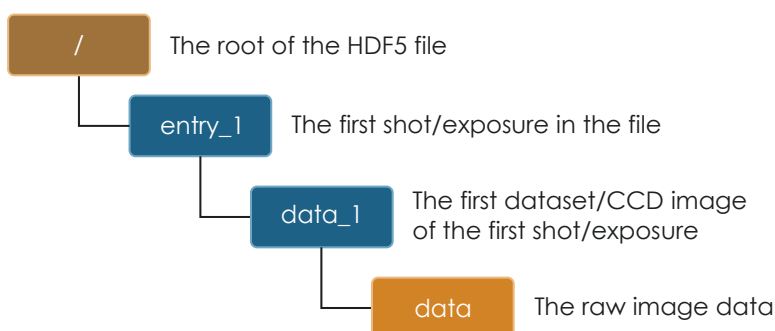
- Groups representing NeXus classes should be named like the class with the `NX` prefix removed and with `_N` added where `N` are consecutive positive integers starting at 1 (e.g. `entry_1` represents the first `NXentry`).
- NeXus class fields should be used as much as possible.

- Each data field must have default units.
- Non default units are defined using the `units` attribute of relevant dataset, just like in NeXus.

### 3 CXI by example

#### 3.1 A minimal CXI file

Figure 1 shows a diagram a minimal CXI file. Each measurement (which is usually an exposure/shot) in a CXI file is stored inside a group named `entry_N` where  $N$  is a positive integer. The first such entry should be named `entry_1` the second `entry_2` and so on. Each entry is a member of the `NXentry` class in NeXus language.

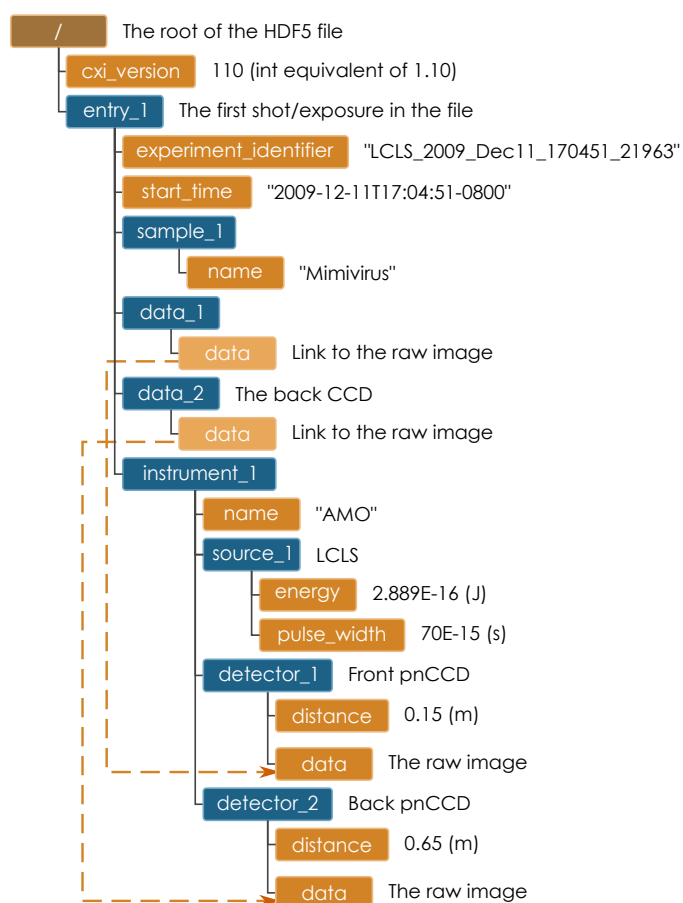


**Figure 1:** Diagram of a minimal CXI file with a single image.

The arrangements of the HDF5 groups is similar to that in a NeXus file. The main difference is that in a NeXus file entries are not identified by name but by the `NX_class` attribute which is set to `NXentry`. In a CXI file we restrict the possible names of a measurement group to only `entry_N` simplifying things by not requiring the attribute. Yet a simple post processing program could take a CXI file and easily convert it into a NeXus file by adding the required attributes. This is the *main design idea* of the CXI format and we will see examples of it in many places.

Each entry can have one or more data groups. In this case we only had one detector so we only have one data group named `data_1`. The raw data is stored in the `data` field of the `data_1` group. As no units are specified the data is assumed to be in “counts” (see 6). Also no experimental data about the experimental conditions is stored.

## 3.2 A typical raw data CXI file

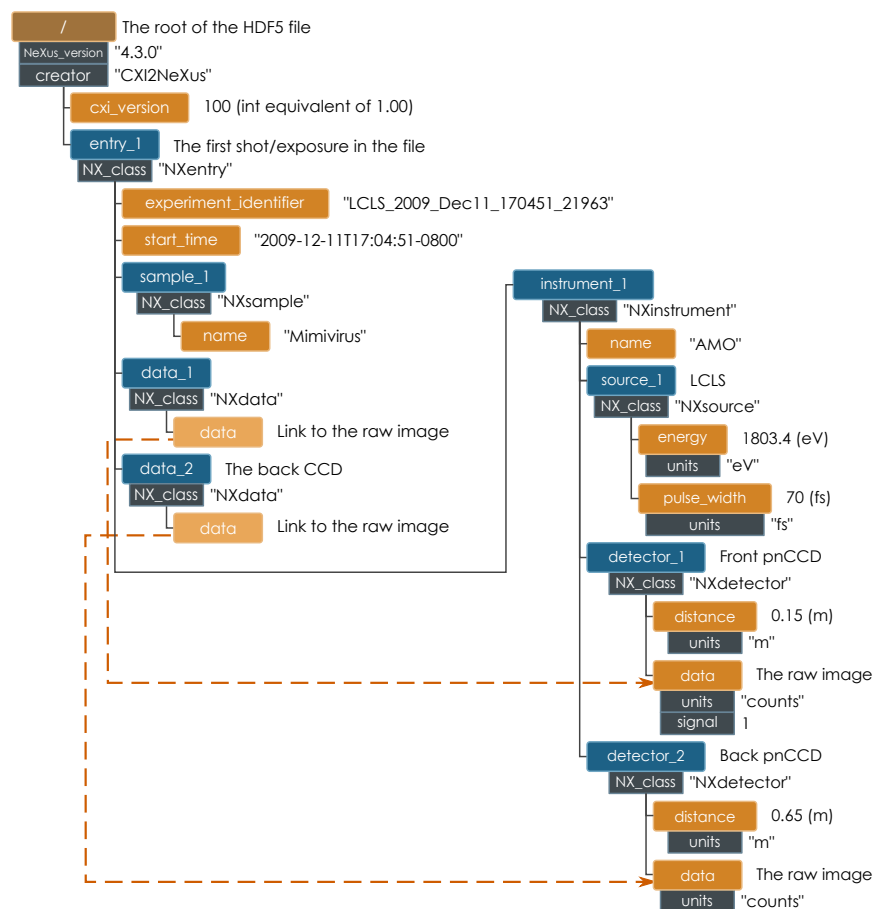


**Figure 2:** Diagram of a typical CXI file for storing raw data from a single shot.

## 4 NeXus compatibility

One of the most appealing features of the CXI format is that while it is relatively simple it can still be unambiguously converted to follow the NeXus format.

Figure 3 shows the conversion of the example in figure 2 to the NeXus format. Note that the only change are the addition of several attributes that are necessary for NeXus. This means that CXI files can be easily converted to NeXus, then edited using NeXus based tools if necessary, and then read back using a CXI based tool.



**Figure 3:** Diagram of a NeXus compatible CXI file for storing raw data from a single shot.



## 5 Datatypes

HDF5 covers a large variety of native datatypes including integers, floating point numbers and character string(includng UTF-8 support). It also takes care of the conversion of datatypes when reading and writing files (eliminating endianness problems for example).

Most of the data should be saved in the same format as it was created/aquired. For example CCD images acquired as 16 bit integers should be saved using the `H5T_NATIVE_SHORT` HDF5 Type.

### 5.1 Complex Numbers

A notable omission of the HDF5 1.8 standard, in which this format is based on, is the lack of a standard way to store complex numbers.

The CXI convention for storing complex numbers is to use a compound data type with two elements named `r` and `i`. The real part of the number should obviously be saved in the element named `r` and the imaginary part in the one named `i`. This follows the convention adopted by PyTables as well as h5py.

Below you can see a C99 code snippet showing how to create a CXI compatible HDF5 compound type for double complex numbers.

Listing 1: Creating a double complex type

```
hid_t complex_id = H5Tcreate(H5T_COMPOUND,
                             sizeof(double complex));
H5Tinsert(complex_id, "r", 0, H5T_NATIVE_DOUBLE);
H5Tinsert(complex_id, "i", sizeof(double), H5T_NATIVE_DOUBLE);
```

## 6 Default units for CXI entries

The default units for all CXI entries are SI base units (see table 1) with *no exceptions*.

**Table 1:** SI (and common derived) base units for different quantities

Quantity	Units	Abbreviation
length	meter	m
mass	kilogram	kg
time	second	s
electric current	ampere	A
temperature	kelvin	K
amount of substance	mole	mol
luminous intensity	candela	cd
frequency	hertz	Hz
force	newton	N
pressure	pascal	Pa
energy	joule	J
power	watt	W
electric potential	volt	V
capacitance	farad	F
electric resistance	ohm	$\Omega$
absorbed dose	gray	Gy
area	square meter	m <sup>2</sup>
volume	cubic meter	m <sup>3</sup>

### 6.1 Angles

Angles are always defined in degrees *not* in radians.

### 6.2 Dates

Dates are always specified according to the [ISO 8601](#). This means for example “1996-07-31T21:15:22+0600”. Note the “T” separating the data from the time and the “+0600” timezone specification.

All of these are mandatory. They derive from the use of ISO 8601 in NeXus. This way compatibility is ensured.

## 7 Memory Layout

All multidimensional arrays must be stored with the fastest changing dimension being the last dimension, and the slowest changing dimension being the first dimension, also known as row major. This is defined in the HDF5 standard.

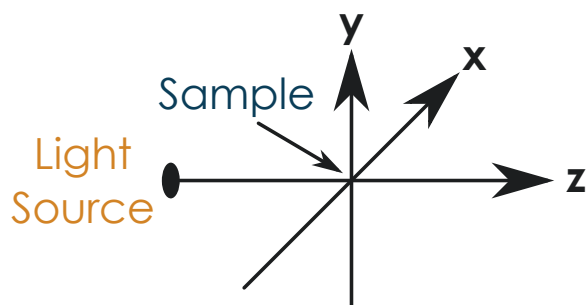
## 8 Geometry

### 8.1 Coordinate System

CXI uses the same coordinate system as NeXus which itself is based on the McStas coordinate system (NeXus User Manual section 2.2.1).

The CXI coordinate system is a right handed system. The  $z$  axis is parallel to the X-ray beam, with the positive  $z$  direction pointing away from the light source, in the downstream direction. This is the *opposite* of the definition of the International Tables for Crystallography, volume G. The  $y$  axis is vertical with the positive direction pointing up, while the  $x$  axis is horizontal completing the right handed system (see Fig. 4).

The origin of the CXI coordinate system is defined by the point where the X-ray beam meets the sample.



**Figure 4:** The coordinate system used by CXI. The intersection of the X-ray beam with the sample define the origin of the system. The  $z$  axis is parallel to the beam and points downstream.

### 8.2 The local coordinate system of objects

For many detectors their location and orientation is crucial to interpret results.

Translations and rotations are used to define the absolute position of each object. But to be able to apply these transformations we need to know what is the origin of the local coordinate system of each object.

Unless otherwise specified the origin should be assumed to be the geometrical center of the object in question. The default orientation of the object should have the longest axis of the object aligned with the x axis, the second longest with the y axis and the shortest with the z axis.

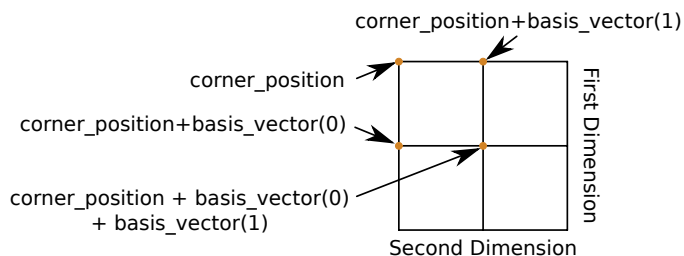
### 8.2.1 The orientation of pixel detectors

The location and orientation of pixel detectors is particularly important for diffraction experiments.

For convenience specific rules have been defined for the coordinate system of pixel detectors. Instead of defining a local origin plus a default orientation the detectors are defined in absolute terms by a corner\_position plus a set of basis\_vectors.

The corner\_position should contain the x, y and z coordinates of the corner of the first data element which corresponds to the corner of the detector. The corner\_position must always be defined.

Figure 5 gives an example.



**Figure 5:** The coordinates of the four corners of the first pixel in a pixel detector.

The basis\_vectors are a matrix containing a set of 3D vectors from the center of the first element to the center of the second element for each dimension of the detector. The number of rows of the matrix (first dimension) is equal to the number of dimensions of data and the number of columns (second dimension) is equal to 3 (the x, y and z position in 3D space).

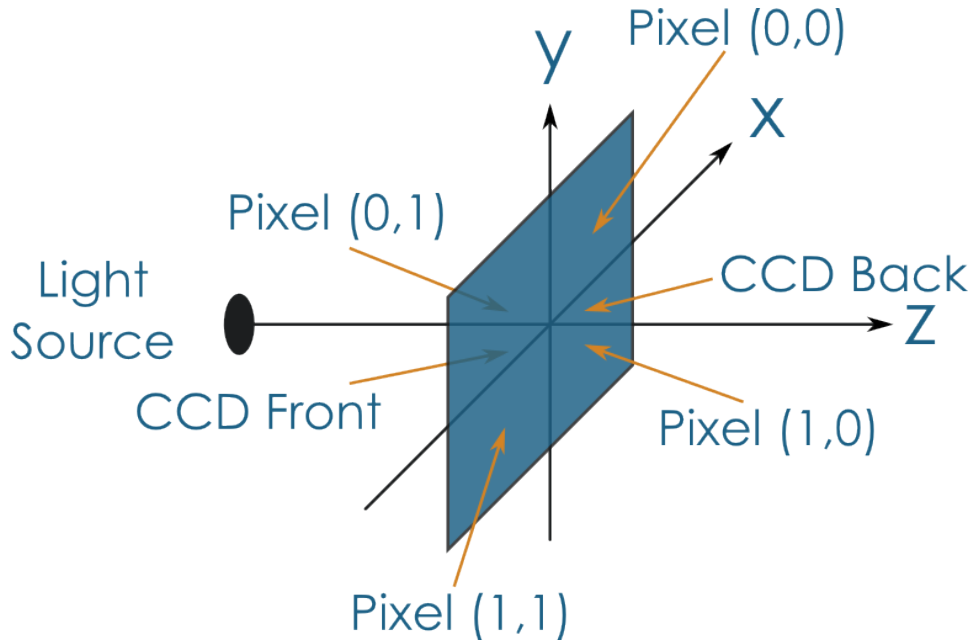
The first row then defines the relative distance between the first two pixels in the first dimension (e.g. position of (1,0) - position of (0,0) for

a 2D array), and the second row elements do the same along the second dimension (e.g. position of (0,1) - position of (0,0) for a 2D array).

If no basis\_vectors are specified they are assumed to be:

$$\begin{bmatrix} 0 & -y\_pixel\_size & 0 \\ -x\_pixel\_size & 0 & 0 \end{bmatrix}$$

This results in the first dimension parallel to the y axis and the second to the x axis. This convention should be used as much as possible as many programs will assume it when displaying data. If more dimensions are required it's strongly encouraged to make the last two dimension correspond to the y and x axis.



**Figure 6:** Pixel locations for a pixel detector with no basis\_vectors defined or rotation applied and with a corner\_position of (pixel width, pixel height, detector distance).

## 9 Scans

While for simplicity it preferable to keep only one image in each Entry class, often there are experiments where many images come together to

form a single dataset. A common case is when one is obtaining multiple images of the same sample at different rotations to perform a tomographic reconstruction. We designate datasets such as these, where one or more variables are changed while images are collected, as *scans*.

In a scan the multiple images are stored in a single 3D or higher dimensional dataset. This has both space and performance advantages, besides making sure the data is kept together. The variables being scanned should always be the first dimensions with the last dimensions reserved for the physical dimensions of the detector. For example in the case of a ptychographic dataset, where the translation of the sample is scanned, the dimensions of the data would be translation, y dimension of the CCD, x dimension of the CCD.

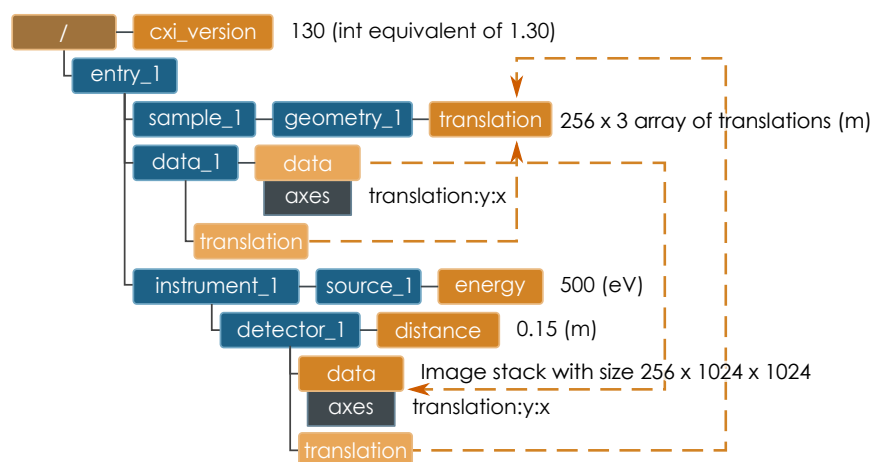
## 9.1 Dimension Scales

When using scans it is crucial to determine what variable corresponds to each dimension. To define this you need to use the *axes attribute*. The `axes` attribute should specify a colon separated list of names of the datasets that correspond to each dimension. The named dataset, or a link to it, should exist in the same group as the field with the `axes` attribute. The exception to this rule are the names “y” and “x” which are assumed to correspond to the number of pixels of the detector in each dimension.

For example in the case of tomography the `axes` attribute of the `data` field could have the value `orientation:y:x` and in the same group there should be a link named `orientation` that links to the orientation field of the sample being imaged. The “x” and “y” are implicitly assumed to correspond to the detector dimensions.

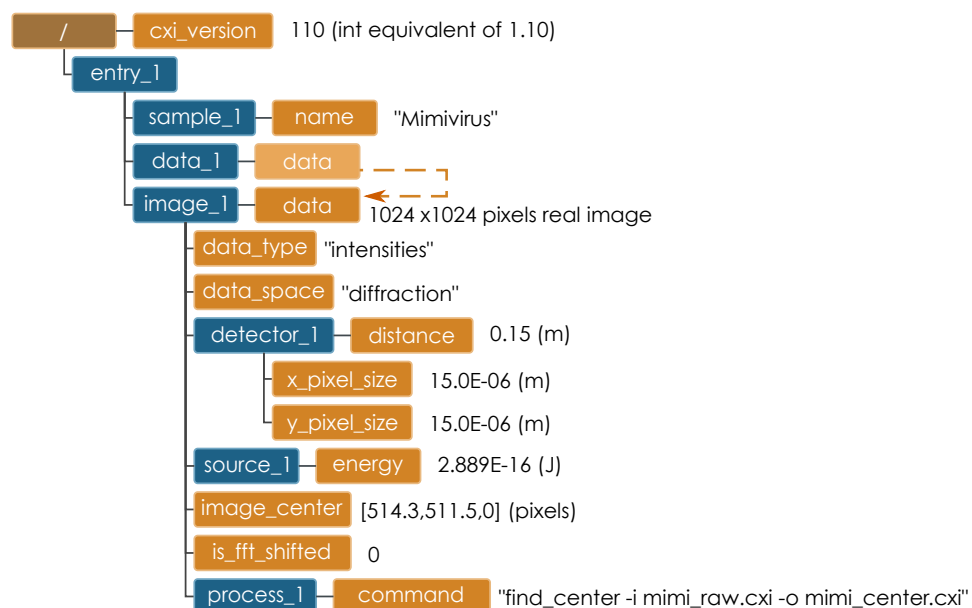
## A More CXI file examples

### A.1 CXI file from a ptychography experiment



**Figure 7:** Diagram of a CXI file with 256 diffraction images recorded on a 1024x1024 pixels CCD detector from a ptychography experiment.

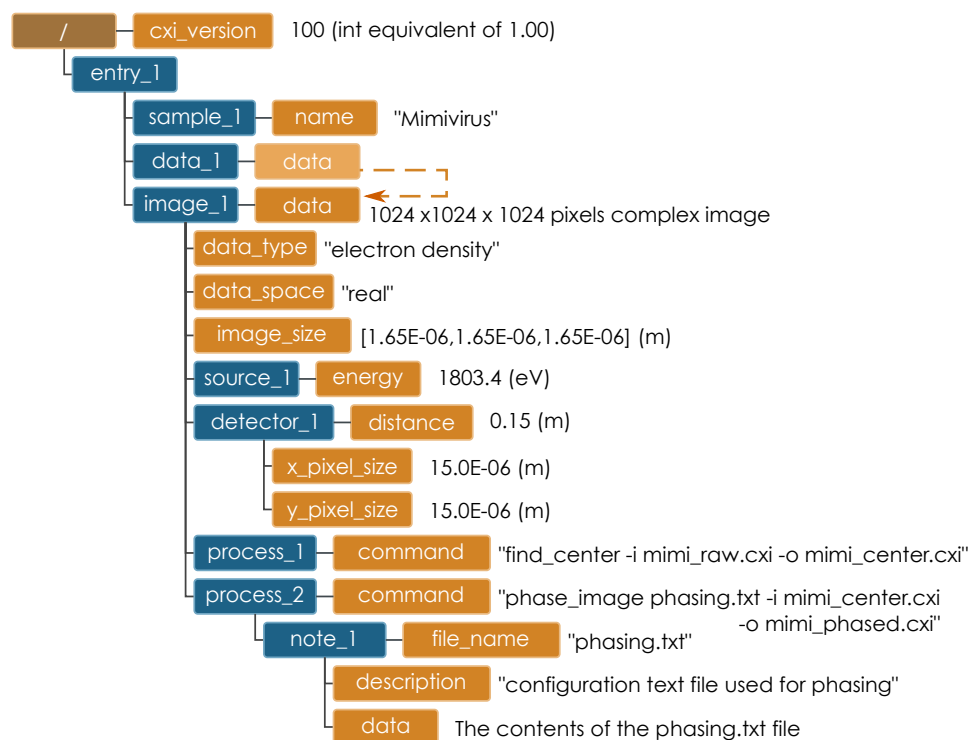
## A.2 CXI file after initial analysis of a diffraction image



**Figure 8:** Diagram of a CXI file of an analysed image from a single particle diffraction experiment.



### A.3 CXI file of a phased 3D image



**Figure 9:** Diagram of a CXI file of a phased 3D image from a single particle diffraction experiment.

## B CXI entries reference

### B.1 Top level (root)

This node represents the top level of the HDF5 file and holds some general information about the file as well as number of entries.

**Table 2:** CXI top level entries

Member	Type	Quantity
<code>cx_i_version</code>	int	version
<code>entry_N</code>	Entry class	
<code>number_of_entries</code>	int	unitless

`cx_i_version` - CXI format version times 100. Version 1.00 would be represented by 100 and 1.5 by 150.

`entry_N` - The measurements recoded in this file.

`number_of_entries` - Total number of entries in the file.

### B.2 Attenuator

This class describes a beamline attenuator used during data collection.

**Table 3:** Attenuator class members

Member	Type	Quantity
<code>distance</code>	float	length
<code>thickness</code>	float	length
<code>attenuator_transmission</code>	float	unitless
<code>type</code>	string	text

`distance` - Distance from sample.

`thickness` - Thickness of attenuator along beam direction.

`attenuator_transmission` - The nominal amount of the beam that gets through (transmitted intensity)/(incident intensity).

`type` - Type or composition of attenuator.

### B.3 Data

This class is a general placeholder for the most important information in each Entry class. It is mandatory that there is at least one Data class in each Entry class. Most data analysis and plotting programs will primarily focus in this class.

**Table 4:** Data class members

Member	Type	Quantity
data	float/complex array	variable
errors	float/complex array	variable

**data** - Most important data values

**errors** - Standard deviations of data values

### B.4 Detector

This class holds information about one of the detectors used during the experiment. Raw data recorded by a detector as well as its position and geometry should be stored in this class.

**Table 5:** Detector class members

Member	Type	Quantity
basis_vectors	float array	length
corner_position	3 floats	length
counts_per_joule	float	unitless
data	float array	variable
data_dark	float array	variable
data_error	float array	variable
data_sum	float	variable
data_white	float array	variable
description	string	text
distance	float	length
geometry_1	<a href="#">Geometry class</a>	
mask	int array	unitless
x_pixel_size	float	length
y_pixel_size	float	length

**basis\_vectors** - A matrix with the basis vectors of the detector data. For more details see 8.2.1.

**corner\_position** - The x, y and z coordinates of the corner of the first data element. For more details see 8.2.1.

**counts\_per\_joule** - Number of counts recorded per each joule of energy received by the detector. The number of incident photons can then be calculated by:

$$\text{number of photons} = \frac{\text{data counts}}{\text{source energy} \times \text{counts per joule}}$$

**data** - Recorded signal values.

**data\_dark** - Image recorded with the shutter closed, used for background subtractions.

**data\_error** - The best estimate of the uncertainty in the data value. Where possible, this should be the standard deviation, which has the same units as the data.

**data\_sum** - Sum of all the elements in the data array. This number is often useful as a cheap measure of data quality.

**data\_white** - Image recorded without the sample, used for background subtractions.

**description** - name/manufacturer/model/etc. information.

**distance** - Closest distance from the detector to the sample.

**geometry\_1** - Position and orientation of the center of mass of the detector. This should only be specified for non pixel detectors. For pixel detectors use **basis\_vectors** and **corner\_position**.

**mask** - Not all the pixels in a detector might have the same value. This 32bit mask makes it possible to distinguish different kinds of pixels.

The following list defines the meaning of each bit when is it set, as well as the names of constants, defined in `cx_i.h`, useful for checkings their values:

All other bits have no standard meaning and can be used for any purpose the user sees fit. More bits will be defined as the format evolves so users are encouraged to use the high bits to avoid collisions.

**x\_pixel\_size** - Width of each detector pixel.

**y\_pixel\_size** - Height of each detector pixel.

**Table 6:** Definition of each bit in the mask

Bit	Meaning	Constant
0x00000001	If set the pixel is valid	CXI_PIXEL_IS_VALID
0x00000002	If set the pixel is saturated	CXI_PIXEL_IS_SATURATED
0x00000004	If set the pixel is hot	CXI_PIXEL_IS_HOT
0x00000008	If set the pixel is dead	CXI_PIXEL_IS_DEAD
0x00000010	If set the pixel is under a shadow	CXI_PIXEL_IS_SHADOWED
0x00000020	If set the pixel is illuminated by parasitic light	CXI_PIXEL_IS_PARASITIC
0x00000040	If set the pixel signal is above the background	CXI_PIXEL_HAS_SIGNAL
0x00000080	If set the pixel does not exist	CXI_PIXEL_DOES_NOT_EXIST
0x00000100	If set the pixel is not exposed	CXI_PIXEL_NOT_EXPOSED

## B.5 Entry

Base CXI class which holds all other classes.

**Table 7:** Entry class members

Member	Type	Quantity
data_ <i>N</i>	Data class	
end_time	string	text
experiment_identifier	string	text
experiment_description	string	text
image_ <i>N</i>	Image class	
instrument_ <i>N</i>	Instrument class	
program_name	string	text
sample_ <i>N</i>	Sample class	
start_time	string	text
title	string	text

**data\_*N*** - Main data collected.

**end\_time** - Ending time of measurement.

**experiment\_identifier** - Unique identifier for the experiment, defined by the facility, possibly linked to the proposals.

**experiment\_description** - Description of the experiment.

**image\_*N*** - Processed images.

**instrument\_*N*** - Instrument used.

**program\_name** - Name of program used to generate this file.

`sample_N` - Sample used.

`start_time` - Starting time of measurement.

`title` - Extended title for entry

## B.6 Geometry

This class holds the general position and orientation of a component.

**Table 8:** Geometry class members

Member	Type	Quantity
<code>orientation</code>	6 floats	unitless
<code>translation</code>	3 floats	length

`orientation` - Dot products between the local and the global unit vectors.

`translation` - The x, y and z components of the translation of the origin of the object relative to the origin of the global coordinate system (the place where the X-ray beam meets the sample).

Only one orientation and one translation is permitted in each geometry class.

The position of the origin of the object should be explicitly defined for each object. If it is not defined it should be assumed to be the center of the object.

The orientation information is stored as direction cosines. The direction cosines will be between the local coordinate directions and the global coordinate directions. The unit vectors in both the local and global coordinates are right-handed and orthonormal.

Calling the local unit vectors  $(x', y', z')$  and the reference unit vectors  $(x, y, z)$  the six numbers will be  $[x' \cdot x, x' \cdot y, x' \cdot z, y' \cdot x, y' \cdot y, y' \cdot z]$  where “ $\cdot$ ” is the scalar dot product (cosine of the angle between the unit vectors).

Notice that this corresponds to the first two rows of the rotation matrix that transforms from the global orientation to the local orientation, namely, the matrix that when applied to  $x'$  produces the vector  $(1, 0, 0)$  and analogously for  $y'$  and  $z'$ . It also corresponds to the first two *columns* of the matrix that when applied to each of the global unit vector  $(x, y, z)$  produces respectively  $(x', y', z')$ . The third row can be recovered by using the fact that the basis vectors are orthonormal.

## B.7 Image

This class should be used to store processed image data. It describes what analysis has been done, as well as holding important information for further image processing. It should not be used for raw data storage, which should be stored in the Detector class.

**Table 9:** Image class members

Member	Type	Quantity
data	float/complex array	variable
data_error	float/complex array	variable
data_space	string	text
data_type	string	text
detector_ <i>N</i>	<a href="#">Detector class</a>	
dimensionality	int	unitless
image_center	3 floats	pixels
image_size	3 floats	(inverse) length
is_fft_shifted	int	unitless
mask	int array	unitless
process_ <i>N</i>	<a href="#">Process class</a>	
reciprocal_coordinates	3 floats array	inverse length
source_ <i>N</i>	<a href="#">Source class</a>	

**data** - The value of the image at each pixel.

**data\_error** - The best estimate of the uncertainty in the data value. Where possible, this should be the standard deviation, which has the same units as the data.

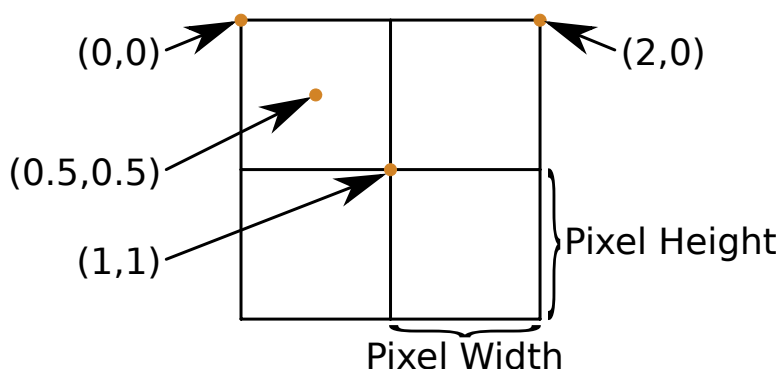
**data\_space** - Specifies if the image lives in real or diffraction (Fourier) space. Only has two valid values: “real” and “diffraction”.

**data\_type** - Defines what the data represents. The following values are allowed: “intensity”, “electron density”, “amplitude”, “unphased amplitude”, “autocorrelation”. “amplitude” implies a phased dataset, while “unphased amplitude” corresponds to the square root of the intensity.

**detector\_*N*** - Link to the detectors used to obtain this image.

**dimensionality** - Number of dimensions of the image. Restricted to 1 2 or 3.

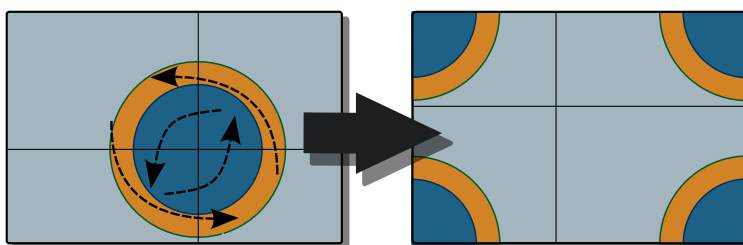
**image\_center** - The location of the zero frequency component on a diffraction image in fractional pixels (see Fig. 10 for the coordinate system convention).



**Figure 10:** Pixel coordinate system for a 2x2 pixels image.

**image\_size** - The width, height and depth of the image. For real space images this corresponds to the length and for reciprocal space ones to the inverse length of the sides of the image.

**is\_fft\_shifted** - If set to 1 the image is assumed to have the quadrants shifted (see Fig. 11).



**Figure 11:** FFT shifting a 2 dimensional image.

**mask** - 32-bit unsigned integer mask specifying the properties of each pixel.

The following list defines the meaning of each bit when is it set, as well as the names of constants, defined in `cxih`, useful for checkings their values:

All other bits have no standard meaning and can be used for any purpose the user sees fit. More bits will be defined as the format evolves so users are encouraged to use the high bits to avoid collisions.

**process\_N** - Processes used to obtain this image. They should be listed in chronological order with the first processed used named `process_1`,



**Table 10:** Definition of each bit in the mask

Bit	Meaning	Constant
0x00000001	If set the pixel is valid	CXI_PIXEL_IS_VALID
0x00000002	If set the pixel is saturated	CXI_PIXEL_IS_SATURATED
0x00000004	If set the pixel is hot	CXI_PIXEL_IS_HOT
0x00000008	If set the pixel is dead	CXI_PIXEL_IS_DEAD
0x00000010	If set the pixel is under a shadow	CXI_PIXEL_IS_SHADOWED
0x00000020	If set the pixel is illuminated by parasitic light	CXI_PIXEL_IS_PARASITIC
0x00000040	If set the pixel signal is above the background	CXI_PIXEL_HAS_SIGNAL
0x00000200	If set the pixel is inside of the reconstruction support	CXI_PIXEL_INSIDE_SUPPORT

the second process\_2 and so on.

**reciprocal\_coordinates** - The diffraction (Fourier) space coordinates of the center of each pixel. Note that the dimension corresponding to the 3 different components should go before the image dimensions. So for example for an image of size [10,20,5] the reciprocal\_coordinates will have size [3,10,20,5].

**source\_N** - Link to the source used to obtain this image.

## B.8 Instrument

Template of instrument descriptions comprising various beamline components. Each component will also be a class defined by its distance from the sample. Negative distances represent beamline components that are before the sample while positive distances represent components that are after the sample. This device allows the unique identification of beamline components in a way that is valid for both reactor and pulsed instrumentation.

Each Instrument instance corresponds to one beamline.

**name** - Name of the instrument.

**attenuator\_N** - The attenuators that are part of the instrument.

**detector\_N** - The detectors that compose the instrument.

**source\_N** - The source used by the instrument.

**Table 11:** Instrument class members

Member	Type	Quantity
name	string	text
attenuator_ <i>N</i>	Attenuator class	
detector_ <i>N</i>	Detector class	
monochromator_ <i>N</i>	Monochromator class	
source_ <i>N</i>	Source class	

## B.9 Monochromator

Define a monochromator used in the instrument.

**Table 12:** Monochromator class members

Member	Type	Quantity
energy	float	energy
energy_error	float	energy

**energy** - Peak of the spectrum that the monochromator selects.

**energy\_error** - Standard deviation of the spectrum that the monochromator selects.

## B.10 Note

This class can be used to store additional information in a CXI file e.g. additional text logs, configuration files, pictures, movies, audio.

**Table 13:** Note class members

Member	Type	Quantity
author	string	text
data	char array	binary/text
date	string	text
description	string	text
file_name	string	text
type	string	text

**author** - Author or creator of note.

**data** - Binary/text note data.

**date** - Date note created/added.

**description** - Title of an image or other details of the note

**file\_name** - Name of original file name if note was read from an external source.

**type** - Mime content type of note data field e.g. image/jpeg, text/plain, text/html.

## B.11 Process

Document an event of data processing, reconstruction, or analysis.

**Table 14:** Process class members

Member	Type	Quantity
command	string	text
comments	string	text
date	string	text
note_ <i>N</i>	Note class	
program	string	text
version	string	text

**command** - Command line used to run the program.

**comments** - Comments related to how the data was processed.

**date** - Date and time of processing in ISO 8601 format.

**note\_*N*** - Notes providing extra information like configuration files used, other inputs required or any other important information.

**program** - Name of the program used.

**version** - Version of the program used.

## B.12 Sample

This class holds basic information about the kind of sample used, its geometry and properties.

**concentration** - Concentration of the sample.

**description** - Description of the sample.

**geometry\_1** - Position and orientation of the center of mass of the sample.

**Table 15:** Sample class members

Member	Type	Quantity
concentration	float	mass/volume
description	string	text
geometry_1	Geometry class	
mass	float	mass
name	string	text
temperature	float	temperature
unit_cell	6 floats	length/angle
unit_cell_group	string	text
thickness	float	length
unit_cell_volume	float	volume

**mass** - Mass of sample.

**name** - Descriptive name of sample.

**temperature** - Sample temperature.

**unit\_cell** - Unit cell parameters (a,b,c  $\alpha, \beta, \gamma$ ).

**unit\_cell\_group** - Crystallographic space group of the crystal in PDB format.

**thickness** - Sample thickness.

**unit\_cell\_volume** - Volume of the unit cell.

### B.13 Source

Class describing the light source being used.

**Table 16:** Source class members

Member	Type	Quantity
energy	float	energy
name	string	text
pulse_energy	float	energy
pulse_width	float	time

**energy** - Energy of each photon.

**name** - The name of the source, for example ALS.

**pulse\_energy** - Sum of the energy of all the photons in the pulse.

`pulse_width` - Duration of the pulse.

## C Changes from previous versions

### C.1 Version 1.3

- Introduced [Scans](#).
- Added the axes attribute to describe dataset dimensions during scans.
- Added the fields data\_dark and data\_white to the [Detector class](#).
- Made the type of all the fields that start with data in the [Detector class](#) float instead of float/complex.
- Merged the Translation and Orientation classes in the [Geometry class](#).
- Correct formula for number of photons in [Detector class](#).
- Updated the examples to correspond to the changes, in particular the ptychography example now makes use if scans.
- Added a name field to the [Source class](#).
- Clarify the description of [Geometry orientation](#).

### C.2 Version 1.2.1

- Added a [Monochromator class](#).

### C.3 Version 1.2

- A new entry has been added to the mask of the [Image class](#) to represent the support used during the reconstruction of an image.

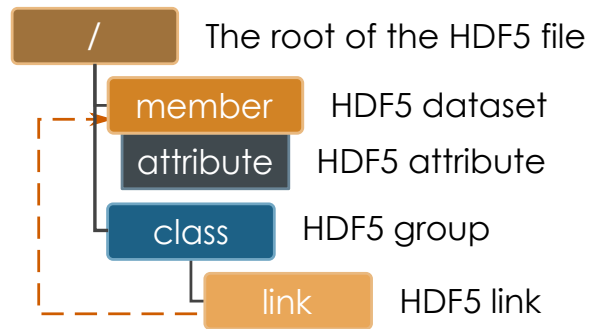
### C.4 Version 1.1

- Add pulse\_energy to the [Source class](#).
- Add data\_sum and change counts\_per\_eV to counts\_per\_joule in the [Detector class](#).
- Added a section(this section) listing all the changes from the previous versions.

## C.5 Version 1.0

- All quantities must now be in SI units, without exceptions.
- The [geometry of CCD detectors](#) is now represented by a series of basis vectors representing the sides of the detector, as well as the absolute corner position, instead of rotations and translations.

## D Diagrams color code



**Figure 12:** Explanation of the color code used in the diagrams



## E Code examples

All the code examples as well as the resulting CXI files are available from [cxidb.org](http://cxidb.org).

### E.1 Creating a minimal CXI file

Listing 2: Creating a minimal CXI file

```
#!/usr/bin/env python

import h5py
import numpy as np
import math

fileName = "minimal.cxi"
# open the HDF5 CXI file for writing
f = h5py.File(fileName, "w")

# create data
x = np.arange(-5, 5, 0.1)
y = np.arange(0, 5, 0.1)
xx, yy = np.meshgrid(x, y)
sinc = np.sin(xx**2+yy**2)/(xx**2+yy**2)

# populate the file with the classes tree
entry_1 = f.create_group("entry_1")
data_1 = entry_1.create_group("data_1")
# write the data
data = data_1.create_dataset("data", data=sinc)

f.close()
```

The resulting file should be equivalent to the one in Fig. 1.

## E.2 Creating a typical raw CXI file

Listing 3: Creating a typical raw CXI file

```
#!/usr/bin/env python

import h5py
import numpy as np
import math

fileName = "typical_raw.cxi"
# open the HDF5 CXI file for writing
f = h5py.File(fileName, "w")
f.create_dataset("cxi_version", data=120)

# create data 1
x = np.arange(-5, 5, 0.1)
y = np.arange(-5, 5, 0.2)
xx, yy = np.meshgrid(x, y)
sinc1 = np.sin(xx**2+yy**2) / (xx**2+yy**2)

# create data 2
x = np.arange(-1, 1, 0.02)
y = np.arange(-1, 1, 0.04)
xx, yy = np.meshgrid(x, y)
sinc2 = np.sin(xx**2+yy**2) / (xx**2+yy**2)

# populate the file with the classes tree
entry_1 = f.create_group("entry_1")
entry_1.create_dataset("experimental_identifier", data=
    "LCLS_2009_Dec11_170451_21963")
entry_1.create_dataset("start_time", data=
    "2009-12-11T17:04:51-0800")
sample_1 = entry_1.create_group("sample_1")
sample_1.create_dataset("name", data="Mimivirus")
instrument_1 = entry_1.create_group("instrument_1")
instrument_1.create_dataset("name", data="AMO")
source_1 = instrument_1.create_group("source_1")
source_1.create_dataset("energy",
    data=2.8893e-16) # in J
source_1.create_dataset("pulse_width",
    data=70e-15) # in s

detector_1 = instrument_1.create_group("detector_1")
detector_1.create_dataset("distance",
    data=0.15) # in meters
detector_1.create_dataset("data", data=sinc1)
```

```
detector_2 = instrument_1.create_group("detector_2")
detector_2.create_dataset("distance",
                          data=0.65) # in meters
detector_2.create_dataset("data", data=sinc2)

data_1 = entry_1.create_group("data_1")
data_1["data"] = h5py.SoftLink('/entry_1/instrument_1/
                               detector_1/data')

data_2 = entry_1.create_group("data_2")
data_2["data"] = h5py.SoftLink('/entry_1/instrument_1/
                               detector_2/data')

f.close()
```

---

The resulting file should be equivalent to the one in Fig. 2.

### E.3 NeXus compatible version of the typical raw CXI file

Listing 4: NeXus compatible version of the typical raw CXI file

```
#!/usr/bin/env python

import h5py
import numpy as np
import math

fileName = "nexus.cxi"
# open the HDF5 CXI file for writing
f = h5py.File(fileName, "w")

f.attrs['NeXus_version'] = "4.3.0"
f.attrs['creator'] = "CXI2NeXus"

f.create_dataset("cxi_version", data=120)

# create data 1
x = np.arange(-5, 5, 0.1)
y = np.arange(0, 5, 0.1)
xx, yy = np.meshgrid(x, y)
sinc1 = np.sin(xx**2+yy**2)/(xx**2+yy**2)

# create data 2
x = np.arange(-1, 1, 0.02)
y = np.arange(0, 1, 0.02)
xx, yy = np.meshgrid(x, y)
sinc2 = np.sin(xx**2+yy**2)/(xx**2+yy**2)

# populate the file with the classes tree
entry_1 = f.create_group("entry_1")
entry_1.create_dataset("experimental_identifier", data=
    "LCLS_2009_Dec11_170451_21963")
entry_1.create_dataset("start_time", data=
    "2009-12-11T17:04:51-0800")
entry_1.attrs['NX_class'] = "NXentry"

sample_1 = entry_1.create_group("sample_1")
sample_1.create_dataset("name", data="Mimivirus")
sample_1.attrs['NX_class'] = "NXsample"

instrument_1 = entry_1.create_group("instrument_1")
instrument_1.create_dataset("name", data="AMO")
instrument_1.attrs['NX_class'] = "NXinstrument"

source_1 = instrument_1.create_group("source_1")
```

```

energy = source_1.create_dataset("energy", data=2.8893e-16) # in
                        J
energy.attrs['units'] = "J"
pulse_width = source_1.create_dataset("pulse_width",
                                      data=70e-15) # in s
pulse_width.attrs['units'] = "s"
source_1.attrs['NX_class'] = "NXsource"

detector_1 = instrument_1.create_group("detector_1")
distance = detector_1.create_dataset("distance",
                                    data=0.15) # in meters
distance.attrs['units'] = "m"
data = detector_1.create_dataset("data", data=sinc1)
data.attrs['signal'] = 1
data.attrs['units'] = "counts"
detector_1.attrs['NX_class'] = "NXdetector"

detector_2 = instrument_1.create_group("detector_2")
distance = detector_2.create_dataset("distance", data=0.65) # in
                        meters
distance.attrs['units'] = "m"
data = detector_2.create_dataset("data", data=sinc2)
data.attrs['units'] = "counts"
detector_2.attrs['NX_class'] = "NXdetector"

data_1 = entry_1.create_group("data_1")
data_1["data"] = h5py.SoftLink('/entry_1/instrument_1/
                             detector_1/data')
data_1.attrs['NX_class'] = "NXdata"

data_2 = entry_1.create_group("data_2")
data_2["data"] = h5py.SoftLink('/entry_1/instrument_1/
                             detector_2/data')
data_2.attrs['NX_class'] = "NXdata"

f.close()

```

---

The resulting file should be equivalent to the one in Fig. 3.

## E.4 3D complex valued Image file

Listing 5: 3D complex valued Image file

```
#!/usr/bin/env python

import h5py
import numpy as np
import math

fileName = "phased_3d.cxi"
# open the HDF5 CXI file for writing
f = h5py.File(fileName, "w")
f.create_dataset("cxi_version", data=120)

# create data
zz, yy, xx = np.mgrid[-5:5:8j, -5:5:12j, -5:5:16j]
sinc = np.sin(xx**2+yy**2+zz**2)/(xx**2+yy**2+zz**2) + \
        1j*np.cos(xx**2+yy**2+zz**2)/(xx**2+yy**2+zz**2)

# populate the file with the classes tree
entry_1 = f.create_group("entry_1")
sample_1 = entry_1.create_group("sample_1")
sample_1.create_dataset("name", data="Mimivirus")
image_1 = entry_1.create_group("image_1")
image_1.create_dataset("data", data=sinc)
image_1.create_dataset("data_type", data="electron density")
image_1.create_dataset("data_space", data="real")
image_1.create_dataset("image_size",
                       data=[1.65e-6, 1.65e-6, 1.65e-6])
source_1 = image_1.create_group("source_1")
source_1.create_dataset("energy", data=2.8893e-16) # in J
detector_1 = image_1.create_group("detector_1")
detector_1.create_dataset("distance",
                          data=0.15) # in meters
detector_1.create_dataset("x_pixel_size",
                          data=15e-6) # in meters
detector_1.create_dataset("y_pixel_size",
                          data=15e-6) # in meters
process_1 = image_1.create_group("process_1")
process_1.create_dataset("command", data="find_center -i"
                        " mimi_raw.cxi -o mimi_center.cxi")
process_2 = image_1.create_group("process_2")
process_2.create_dataset("command", data="phase_image"
                        " phasing.txt -i mimi_center.cxi"
                        " -o mimi_phased.cxi")
note_1 = process_2.create_group("note_1")
note_1.create_dataset("file_name", data="phasing.txt")
```

```
note_1.create_dataset("description", data="configuration text"
                      " file used for phasing")
note_1.create_dataset("data", data='algorithm = "HIO"')

data_1 = entry_1.create_group("data_1")
data_1["data"] = h5py.SoftLink('/entry_1/image_1/data')

f.close()
```

---

The resulting file should be equivalent to the one in Fig. 9.

## E.5 Spartan viewer for CXI files

Listing 6: Spartan viewer for CXI files

```
#!/usr/bin/python
import h5py
import sys
from PyQt4 import QtGui, QtCore, Qt
import matplotlib.pyplot as plt
from operator import mul
import numpy
import signal
import scipy.interpolate
import scipy.ndimage

signal.signal(signal.SIGINT, signal.SIG_DFL)

def onclick(event):
    z = plt.gca().get_images()[0].get_array()
    x = numpy.arange(0, z.shape[0], 1)
    y = numpy.arange(0, z.shape[1], 1)
    v = scipy.ndimage.map_coordinates(z, [[event.ydata], [event
        .xdata]], order=1)
    print 'xdata=%f, ydata=%f value=%e'%(
        event.xdata, event.ydata, v[0])

class Viewer(QtGui.QMainWindow):
    def __init__(self):
        QtGui.QMainWindow.__init__(self)
        self.tree = QtGui.QTreeWidget(self)
        self.setCentralWidget(self.tree)
        self.buildTree()
        self.tree.itemClicked.connect(self.handleClick)
    def handleClick(self, item, column):
        if item.text(column) == "Click to display":
            data = self.datasets[str(item.text(2))]
            fig = plt.figure()
            ax = fig.add_axes([0, 0, 1, 1])
            if numpy.iscomplexobj(data):
                data = numpy.abs(data)
            if len(data.shape) == 1:
                plt.plot(data)
            else:
                ax.imshow(data)
            cid = fig.canvas.mpl_connect('
                button_press_event', onclick)
    def buildTree(self):
        self.datasets = {}
        self.tree.setColumnCount(2)
```



```
self.f = h5py.File(sys.argv[1], "r")
item = QtGui.QTreeWidgetItem(QtCore.QStringList("/"))
self.tree.addTopLevelItem(item)
self.buildBranch(self.f, item)
def buildBranch(self, group, item):
    for g in group.keys():
        lst = QtCore.QStringList(g)
        if(isinstance(group[g], h5py.Group)):
            child = QtGui.QTreeWidgetItem(lst)
            self.buildBranch(group[g], child)
            item.addChild(child)
        else:
            if(not group[g].shape or reduce(mul, group[g]
                                           ].shape) < 10):
                lst.append(str(group[g] [ () ]))
            else:
                lst.append("Click to display")
                lst.append(group[g].name)
                self.datasets[group[g].name] = group[g]
            item.addChild(QtGui.QTreeWidgetItem(lst))

app = QtGui.QApplication(sys.argv)
aw = Viewer()
aw.show()
sys.exit(app.exec_())
```

---

This simple program provides a way to view CXI files. Does not support 3D or complex valued data. To use it pass the name of the file you wish to view as a command line argument.