The Scientific Data Exchange Reference Guide

Includes implementation for x-ray tomography

http://www.aps.anl.gov/DataExchange

Version 0.9.1

April 26, 2013

Table 1: Version history

Version	Date	Notes
0.8.9	Feb. 25, 2012	de carlo: first document draft.
0.9.0	May 25, 2012	saunders: Extensive reworking of document structure and some definitions.
0.9.1	April 20, 2012	de carlo: added Python examples and corrected minor tomography definitions. Changed provenance definition.

Contents

1 Introduction

This document is a complete reference to the Data Exchange file format, including documented implementations for various beamline techniques. Briefly, Data Exchange is a set of guidelines for storing scientific data and metadata in a Hierarchical Data Format 5 (HDF5) file (http://www.hdfgroup.org/HDF5).

This reference guide describes the basic design principles, examples of their application, a core reference for guidelines common to most uses, and coding examples. The guide ends with a section devoted to each known implementation for a particular beamline technique.

2 The Design of Data Exchange

For various reasons, many x-ray techniques developed at synchrotron facilities around the world are unable to directly compare results due to their inability to exchange data and software tools. The aim of Data Exchange is to define a simple file format offering few basic rules and allowing each community to extend and add technique specific components. The goal is to provide extensibility in defining data, meta data and provenance information in a simple way that can be easily adopted by various x-ray techniques.

The Data Exchange format is implemented using Hierarchical Data Format 5 (HDF5), which offers platform-independent binary data storage with optional compression, hierarchical data ordering, and self-describing tags so that one can examine a HDF5 file's contents with no knowledge of how the file writing program was coded.

The aim and the scope of Data Exchange is very similar to the Coherent X-ray Imaging Data Bank file format (CXI), so whenever possible we will use the same conventions, name tags, and reference system. This document is using a similar diagram definition set by Filipe R. N. C. Maia in "CXI file format" (http://cxidb.org/cxi.html), with a few minor modifications for Data Exchange definitions.

The core principle of Data Exchange is that it must be simple enough that it is not necessary to use a support library beyond core HDF5. The simplicity of Data Exchange makes it easy for anyone to either look at an example file using h5dump or HDFView, or to look at example code in language X, and then create their own read and write routines in language Y.

The simplest Data Exchange file provides information sufficient to share a multidimensional data array. In this simplest form, Data Exchange implements only one "exchange" group. The "exchange" definition is designed to allow for simple exchange of images, spectra, and other forms of beamline detector data with a minimum of fields. This definition is essentially a technique-agnostic format for exchanging data with others. Data Exchange is also designed to be extended to include technique-specific data and metadata. This is achieved by providing optional, but clearly defined, metadata components to the base definition.

2.1 HDF5

The HDF5 format is the basis of the Data Exchange format. Data Exchange, like CXI, is not a completely new file format, but simply a set of rules designed to create HDF5 files with a common structure 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 fulfillment of the Data Exchange requirements, i.e. simplicity, flexibility and extensibility. HDF5 version

1.8 or higher is required as previous versions don't support all features required by Data Exchange.

2.2 Data types

Data Exchange uses the same CXI convention for data types as defined at `http://cxidb.org/cxi.html` using HDF5 native datatypes. The data should be saved in the same format as it was created/acquired. For example CCD images acquired as 16 bit integers should be saved using the `H5T_NATIVE_SHORT` HDF5 type. In this way all cross platform big-little endian issues reading and writing files are eliminated.

2.3 Root Level Structure

While HDF5 gives great flexibility in data storage, straightforward file readability and exchange requires adhering to an agreed-upon naming and organizational convention. To achieve this goal, Data Exchange adopts a layered approach by defining a set of *mandatory* and `optional` fields.

The general structure of a Data Exchange file is shown in table **??**. The most basic file must have an "implements" string, and an "exchange" group at the root level/group of the HDF5 file. Optional "measurement" and "provenance" groups are also defined. Beyond this, additional groups may be added to meet individual needs, with guidelines suggesting the best structure.

Table 2: Data Exchange Top Level Members

Member	Type	Example
implements	string dataset	"exchange:measurement:provenance"
exchange	group	
`measurement`	group	
`provenance`	group	

implements

Mandatory scalar string dataset in the root of the HDF5 file whose value is a colon separated list that shows which components are present in the file. All components listed in the *implements* string are to be groups placed in the HDF5 file at the root level/group. In a minimal Data Exchange file, the only mandatory item in this list is *exchange*. A more general Data Exchange file also contain `measurement` and possibly `provenance`, in which case the implements string would be: "*exchange*: `measurement`: `provenance`"

exchange

Mandatory group containing one or more arrays that represent the most basic version of the data, such as raw or normalized optical density maps or a

elemental signal map. *Exchange_N* is used when more than one core dataset or derived datasets are saved in the file. The *exchange* implementation for specific techniques are defined in separate sections in the Reference Guide.

`measurement`
: Optional group containing the measurement made on the sample. `Measurement` contains information about the `sample` and the `instrument`. `Measurement_`*N* is used when more than one measurement is stored in the same file.

`provenance`
: Optional group containing information about the status of each processing step.

In a Data Exchange file, each dataset has a unit defined using the `units` attribute. `units` is not mandatory - if omitted, the default unit as defined in Appendix **??** is used.

The detailed rules about how to store datasets within the exchange group are best shown through examples in the next section. Detailed reference information can be found in the Data Exchange Core Reference section.

3 Data Exchange by Example

The examples in this section show how one can store data for imaging experiments using the Data Exchange format. It is general enough, however, to show how Data Exchange can be extended or adapted to other techniques. These examples are meant to give a flavor for our approach. A complete reference to the core structure can be found in Section **??**. Technique specific extensions to the core structure can be found at the end of the Reference Guide.

3.1 Diagram color code

All the diagrams in this section follow the color conventions shown in Figure **??**. The basic elements are HDF5 datasets, attributes, and groups. We also support internal references to elements in the file by a simple scalar string that holds the path of the dataset within the file. On the diagram, this is shown as a reference dataset that points to the referred-to dataset. Note that we use this mechanism rather than HDF5 hard or soft links

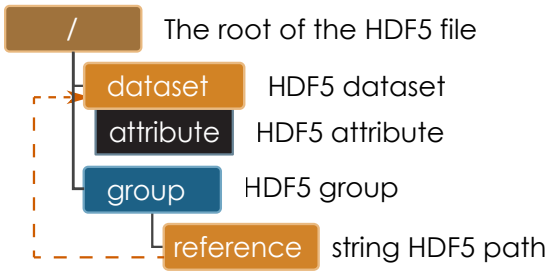

Figure 1: Explanation of the color code used in the diagrams

3.2 A minimal Data Exchange file for imaging

Figures **??** shows a diagram of a minimal Data Exchange file to store a single projection image. It is strongly encouraged that all datasets shall have a units attribute. The axes of the dataset are not specified in this minimal case, and can be assumed to be x and y with a zero-based integer sequence, or more simply, pixels.

3.3 Storing and describing a multidimensional dataset

A multidimensional dataset should be described as fully as possible, with units for the dataset as well as dimension descriptors (that also have units defined). There are also additional descriptive fields available such as title and description. The order of dimensions in the dataset should put the slowest changing dimension first, and the fastest changing dimension last.

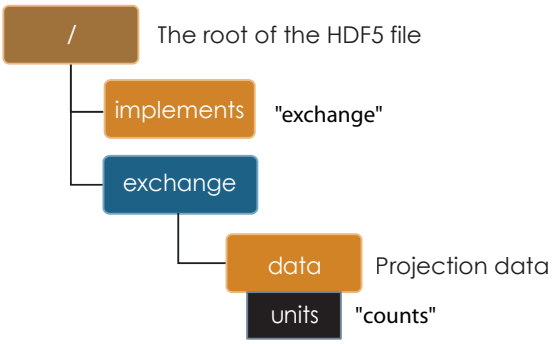

Figure 2: Diagram of a minimal Data Exchange file for a single image.

It is strongly encouraged that all datasets have a units attribute. The string value for units should preferably be an SI unit, however well understood non-SI units are acceptable, in particular ”degrees”. The units strings should conform to those defined by UDUNITS at http://www.unidata.ucar.edu/software/udunits. While UDUNITS is a software package, it contains simple XML files that describe units strings and acceptable aliases.

The axes of a multidimensional dataset are described through the use of additional one-dimensional datasets (dimension descriptors), one for each axis in the main dataset. Take for example a 3-dimensional cube of images, with axes of x, y, and z where z represents the angle of the sample when each image was taken. There should be 3 additional one-dimensional datasets called x, y, and z where x and y contain an integer sequence, and z contains a list of angles. X and y have units of ”counts” and z has units of ”degrees”. To simplify, it is acceptable to omit x and y, since the default interpretation will always be an integer sequence.

The dimension descriptors (x, y, and z) can be associated with the main dataset through two mechanisms. The HDF5 libraries contain a function call H5DSattach_scale to ”attach” a dimension descriptor dataset to a given dimension of the main dataset. HDF5 takes care of entering several attributes in the file that serve to keep track of this association. If the particular programming language you work in does not support this HDF5 function, then you can instead add a string attribute to your main dataset called axes. The axes attribute is simply a colon separated string naming the dimension descriptor datasets in order, so ”z:y:x” in this case. Additional examples below show this in action.

3.4 Storing projections, dark fields, and white fields

A tomographic data set consists of a series of projections, dark and white field images. The dark and white fields must have the same projection image dimensions and can be collected at any time before, after or during the projection data collection. The angular position of the tomographic rotation axis, theta, can be used to keep track of when the dark and white images are collected. These

examples show projection, dark, and white images saved in three 3D arrays as shown in Figure **??** and **??** using, by default, the natural HDF5 order of the a multidimensional array (rotation axis, ccd y, ccd x), i.e. with the fastest changing dimension being the last dimension, and the slowest changing dimension being the first dimension. If using the default dimension order, the axes attribute ”theta:y:x” can be omitted. The `axes` attribute is mandatory if the 3D arrays use a different axes order. This could be the case when, for example, the arrays are optimized for sinogram read (`axes` = ”y:theta:x”). As no units are specified the data is assumed to be in “counts” with the axes (x, y) in pixels.

If the positions of the rotation axis for each projection, dark, and white images are not specified via theta dimension scale datasets, it is assumed that the raw projections are taken at equally spaced angular intervals between 0 and 180 degree, with white and dark field collected at the same time before or after the projection data collection.

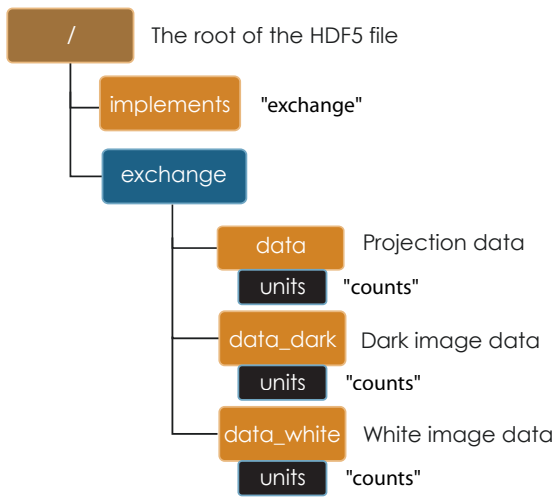

Figure 3: Diagram of a minimal Data Exchange file for a single tomographic data set including raw projections, dark, and white fields.

3.5 A typical Data Exchange file for tomography

A series of tomographic data sets are typically collected changing the instrument status (energy, detector or optics position) or changing the sample status (position, environment etc.). Figure **??**, **??** and **??** show the content of files changing the sample temperature, the x-ray source energy and detector-sample distance.

3.5.1 Sample Temperature Scan

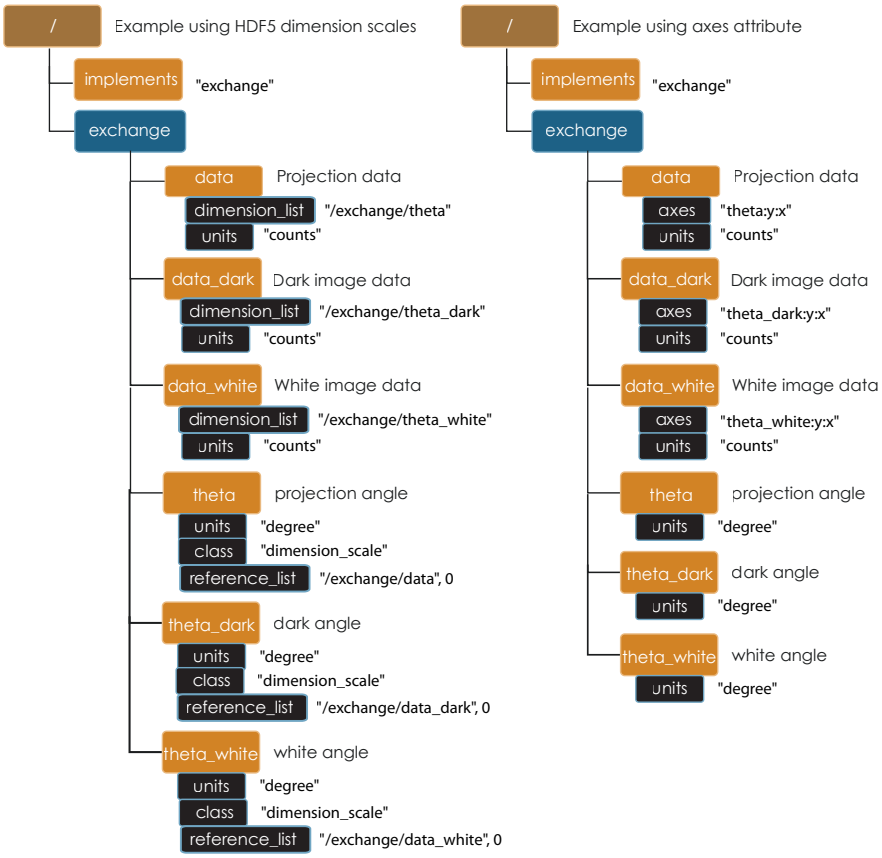

Figure 4: Diagram of a single tomographic data set including raw projections, dark and white fields. In this case, there are additional dimension descriptor datasets theta, theta_dark, and theta_white that contain the positions of the rotation axis for each projection, dark, and white image. The lefthand example shows this as it would appear using the HDF5 H5DSattach_scale function. The righthand example shows this as it would appear by manually adding an axes attribute (for cases where H5DSattach_scale is unavailable).

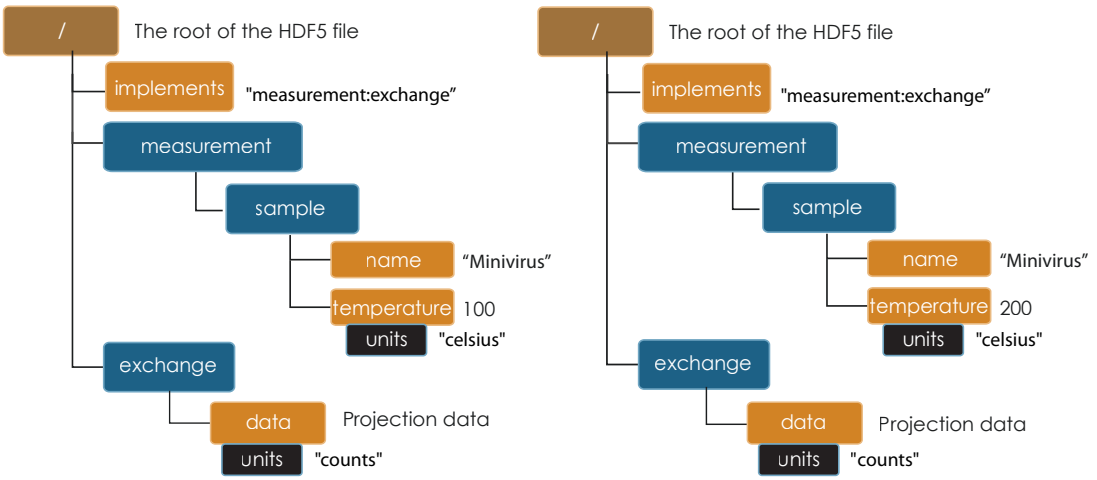

Figure 5: Diagram of two tomographic data sets taken at two different sample temperatures (100 and 200 celsius).

3.5.2 X-ray Energy Scan

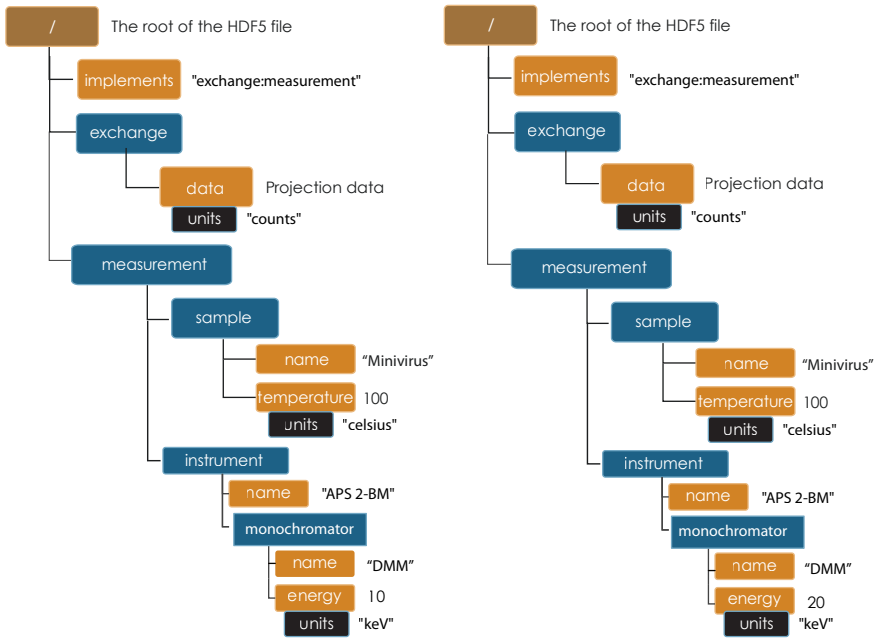

Figure 6: Diagram of two tomographic data sets taken at two different energy (10 and 20 keV).

3.5.3 Detector-sample Distance Scan

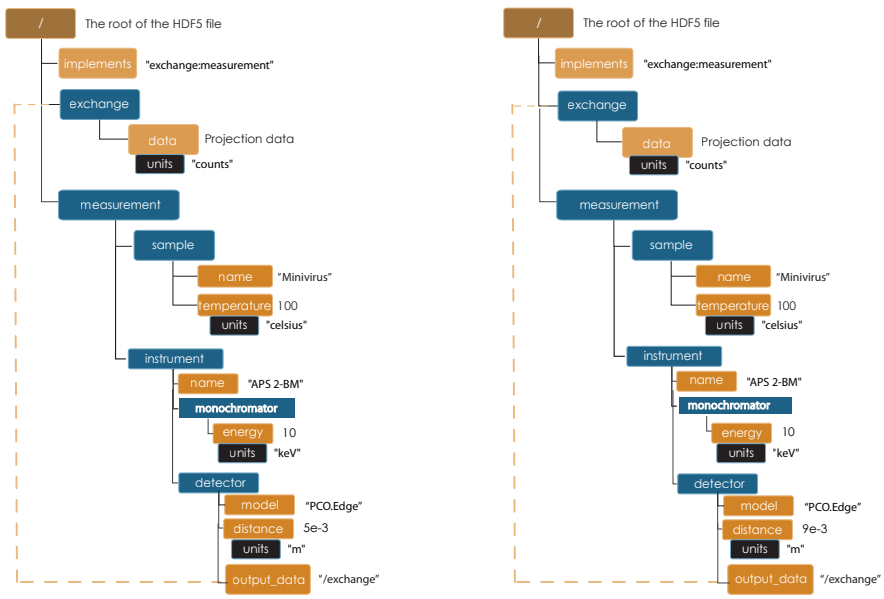

Figure 7: Diagram of two tomographic data sets collected with two different detector-sample distances (5 and 9 mm). Note the use of `output_data` dataset to associate the detector with the exchange group generated from the acquisition.

3.5.4 Series of Tomographic Measurements

A series of tomographic measurements, when relevant, can be stored in the same file appending $_N$ to the measurement tag. In nano tomography experiments, for example, the detector field of view is often smaller than the sample. To collect a complete tomographic data set, it is necessary to raster the sample across the field of view moving its x and y location. Figure **??** shows a file from a nano tomography experiment when the sample rasters through the field of view.

There are limits to this approach, as one clearly does not want to have hundreds of measurement groups in a file (or multiple files) where most of the metadata is the same. For measurements where there are many ”positioner” values (aka a ”scan”), it is more sensible to add dimension(s) to the exchange dataset, and describe the ”positioner” values as dimension scales. This is a judgement left to the user.

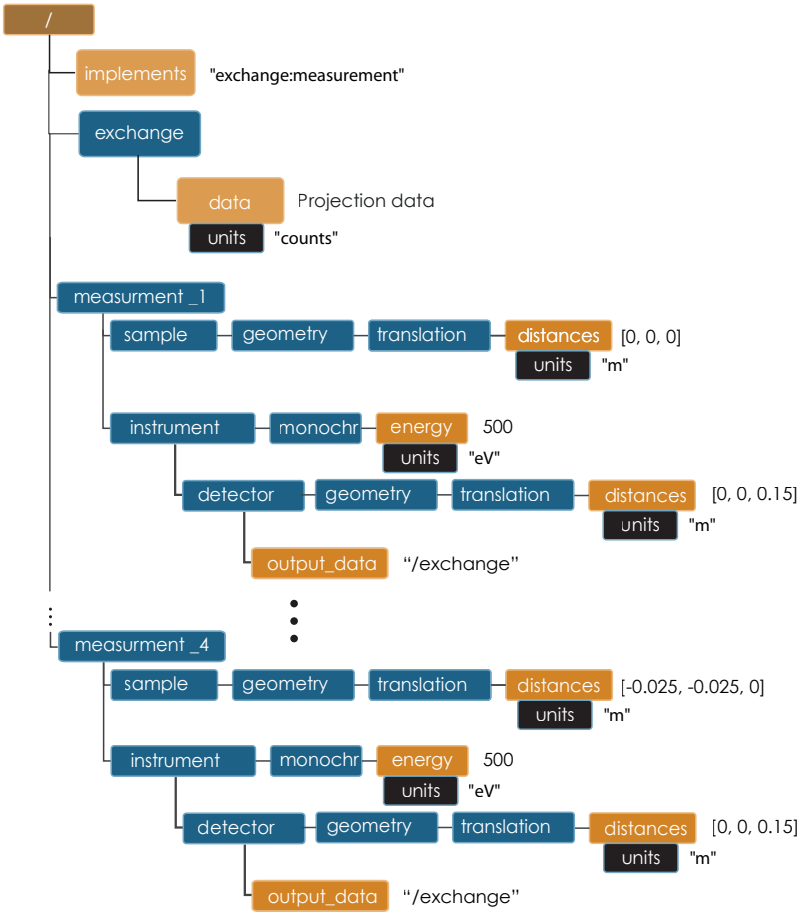

Figure 8: Diagram of a file with 4 tomographic data sets from a nano tomography experiment.

4 Data Exchange Core Reference

4.1 Top level (root)

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

Table 3: Top Level Members

Member	Type	Example
implements	string dataset	"exchange:measurement:provenance"
exchange_N	group	
`measurement_N`	group	
`provenance`	group	

implements
: A colon separated list that shows which components are present in the file. The only *mandatory* component is *exchange*. A more general Data Exchange file also contains `measurement` and `provenance` information, if so these will be declared in implements as "exchange:measurement:provenance"

exchange_N
: The data taken from measurements or processing. Dimension descriptors within the group may also serve to describe "positioner" values involved in a scan.

`measurement_N`
: Description of the sample and instrument as configured for the measurement. This group is appropriate for relatively static metadata. For measurements where there are many "positioner" values (aka a "scan"), it is more sensible to add dimension(s) to the exchange dataset, and describe the "positioner" values as dimension scales rather than record the data via multiple matching measurement_N and exchange_N groups. This is a judgement left to the user.

`provenance`
: The Provenance group describes all process steps that have been applied to the data.

4.2 Exchange

The exchange group is where scientific datasets reside. This group contains one or more array datasets containing n-dimensional data and optional descriptions of the axes (dimension scale datasets). Exactly how this group is used is dependent on the application, however the general idea is that one exchange group contains one cohesive dataset. If, for example, the dataset is processed into some other form, then another exchange group is used to store the derived data.

Multiple exchange groups are numbered consecutively as exchange_N. At a minimum, each exchange group should have a primary dataset named data. The title is optional.

Table 4: Exchange Group Members

Member	Type	Example
`title`	string dataset	"tomography projections"
data	array dataset	n-dimensional dataset

`title`
: Descriptive title for data dataset.

data
: The primary scientific dataset. Additional related datasets may have any arbitrary name. Each dataset should have a units and description attribute. Discussion of dimension descriptors and optional axes attribute is covered in Section **??**.

Table 5: data attributes

Attribute	Type	Example
`description`	string attribute	"transmission"
`units`	string attribute	"counts"

4.2.1 Storing and describing a multidimensional dataset

A multidimensional dataset should be described as fully as possible, with units for the dataset as well as dimension descriptors (that also have units defined). There are also additional descriptive fields available such as title and description. The order of dimensions in the dataset should put the slowest changing dimension first, and the fastest changing dimension last.

It is strongly encouraged that all datasets have a units attribute. The string value for units should preferably be an SI unit, however well understood non-SI units are acceptable, in particular "degrees". The units strings should conform

to those defined by UDUNITS at http://www.unidata.ucar.edu/software/udunits. While UDUNITS is a software package, it contains simple XML files that describe units strings and acceptable aliases.

The axes of a multidimensional dataset are described through the use of additional one-dimensional datasets (dimension descriptors), one for each axis in the main dataset. Take for example a 3-dimensional cube of images, with axes of x, y, and z where z represents the angle of the sample when each image was taken. There should be 3 additional one-dimensional datasets called x, y, and z where x and y contain an integer sequence, and z contains a list of angles. X and y have units of ”counts” and z has units of ”degrees”. To simplify, it is acceptable to omit x and y, since the default interpretation will always be an integer sequence.

The dimension descriptors (x, y, and z) can be associated with the main dataset through two mechanisms. The HDF5 libraries contain a function call to ”attach” a dimension descriptor dataset to a given dimension of the main dataset. HDF5 takes care of entering several attributes in the file that serve to keep track of this association. If the particular programming language you work in does not support this HDF5 function, then you can instead add a string attribute to your main dataset called axes. The axes attribute is simply a colon separated string naming the dimension descriptor datasets in order, so ”z:y:x” in this case.

4.3 Measurement

This group holds sample and instrument information. These groups are designed to hold relatively static data about the sample and instrument configuration at the time of the measurement. Rapidly changing ”positioner” values (aka scan) are better represented in the exchange group dataset.

There is a geometry group common to many of the subgroups under measurement. The intent is to describe the translation and rotation (orientation) of the sample or instrument component relative to some coordinate system. Since we believe it is not possible to determine all possible uses at this time, we leave the precise definition of geometry up to the technique. We do encourage the use of separate translation and orientation subgroups within geometry. As such, we do not describe geometry further here.

Table 6: Measurement Group Members

Member	Type	Example
sample	group	
instrument	group	

`sample`
: The sample measured.

`instrument`
: The instrument used to collect this data.

4.3.1 Sample

This group holds basic information about the sample, its geometry, properties, the sample owner (user) and sample proposal information. While all these fields are optional, if you do intend to include them they should appear within this parentage of groups.

`name`
: Descriptive name of the sample.

`description`
: Description of the sample.

`preparation_date`
: Date and time the sample was prepared.

`chemical_formula`
: Sample chemical formula using the CIF format.

Table 7: Sample Group Members

Member	Type	Example
name	string dataset	"cells sample 1"
description	string dataset	"malaria cells"
preparation_date	string dataset (ISO 8601)	"2012-07-31T21:15:22+0600"
chemical_formula	string dataset (abbr. CIF format)	"(Cd 2+)3, 2(H2 O)"
mass	float dataset	0.25
concentration	float dataset	0.4
environment	string dataset	"air"
temperature	float dataset	25.4
temperature_set	float dataset	26.0
pressure	float dataset	101325
thickness	float dataset	0.001
position	string dataset	"2D" APS robot coord.
geometry	group	
experiment	group	
experimenter	group	

`mass`
: Mass of the sample.

`concentration`
: Mass/volume.

`environment`
: Sample environment.

`temperature`
: Sample temperature.

`temperature_set`
: Sample temperature set point.

`pressure`
: Sample pressure.

`thickness`
: Sample thickness.

`position`
: Sample position in the sample changer/robot.

`geometry`
: Sample center of mass position and orientation.

`experiment`
: Facility experiment identifiers.

`experimenter`
: Experimenter identifiers.

4.3.1.1 Geometry

This class holds the general position and orientation of a component. We do not define this further here.

Table 8: Geometry Group Members

Member	Type	Example
translation	group	
orientation	group	

translation
: The position of the object with respect to the origin of your coordinate system.

orientation
: The rotation of the object with respect to your coordinate system.

4.3.1.2 Experiment

This provides references to facility ids for the proposal, scheduled activity, and safety form.

Table 9: Experiment Group Members

Member	Type	Example
proposal	string dataset	"1234"
activity	string dataset	"9876"
safety	string dataset	"9876"

`proposal`
: Proposal reference number. For the APS this is the General User Proposal number.

`activity`
: Proposal scheduler id. For the APS this is the beamline scheduler activity id.

`safety`
: Safety reference document. For the APS this is the Experiment Safety Approval Form number.

4.3.1.3 Experimenter

Description of a single experimenter. Multiple experimenters can be represented through numbered entries such as experimenter_1, experimenter_2.

Table 10: Experimenter Group Members

Member	Type	Example
name	string dataset	"John Doe"
role	string dataset	"Project PI"
affiliation	string dataset	"University of California, Berkeley"
address	string dataset	"EPS UC Berkeley CA 94720 4767 USA"
phone	string dataset	"+1 123 456 0000"
email	string dataset	"johndoe@berkeley.edu"
facility_user_id	string dataset	"a123456"

`name` User name.

`role` User role.

`affiliation` User affiliation.

`address` User address.

`phone` User phone number.

`email` User e-mail address

`facility_user_id` User badge number

4.3.2 Instrument

The instrument group stores all relevant beamline components status at the beginning of a measurement. While all these fields are optional, if you do intend to include them they should appear within this parentage of groups.

Table 11: Instrument Group Members

Member	Type	Example
`name`	string dataset	"XSD/2-BM"
source	group	
shutter_N	group	
attenuator_N	group	
monochromator	group	
detector_N	group	

`name`
: Name of the instrument.

`source`
: The source used by the instrument.

`shutter_N`
The shutter(s) used by the instrument.

`attenuator_N`
The attenuators that are part of the instrument.

`monochromator`
The monochromator used by the instrument.

`detector_N`
The detectors that compose the instrument.

4.3.2.1 Source

Class describing the light source being used.

Table 12: Source Group Members

Member	Type	Example
name	string dataset	"APS"
datetime	string dataset (ISO 8601)	"2011-07-15T15:10Z"
beamline	string dataset	"2-BM"
current	float dataset	0.094
energy	float dataset	4.807e-15
pulse_energy	float dataset	1.602e-15
pulse_width	float dataset	15e-11
mode	string dataset	"TOPUP"
beam_intensity_incident	float dataset	55.93
beam_intensity_transmitted	float dataset	100.0
geometry	group	

`name`
Name of the facility.

`datetime`
Date and time source was measured.

`beamline`
Name of the beamline.

`current`
Electron beam current (A).

`energy`
Characteristic photon energy of the source (J). For an APS bending magnet this is 30 keV or 4.807e-15 J.

`pulse_energy`
Sum of the energy of all the photons in the pulse (J).

`pulse_width`
: Duration of the pulse (s).

`source`
: Beam mode: TOPUP.

`beam_intensity_incident`
: Incident beam intensity in (photons per s).

`beam_intensity_transmitted`
: Transmitted beam intensity (photons per s).

4.3.2.2 Shutter

Class describing the shutter being used.

Table 13: Shutter Group Members

Member	Type	Example
name	string dataset	"Front End Shutter 1"
status	string dataset	"OPEN"
geometry	group	

`name`
: Shutter name.

`status`
: "OPEN" or "CLOSED" or "NORMAL"

4.3.2.3 Attenuator

This class describes the beamline attenuator(s) used during data collection. If more than one attenuators are used they will be named as attenuator_1, attenuator_2 etc.

Table 14: Attenuator Group Members

Member	Type	Example
thickness	float dataset	1e-3
attenuator_transmission	float dataset	unit-less
type	string dataset	"Al"
geometry	group	

`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.

4.3.2.4 Monochromator

Define the monochromator used in the instrument.

Table 15: Monochromator Group Members

Member	Type	Example
type	string dataset	"Multilayer"
energy	float dataset	1.602e-15
energy_error	float dataset	1.602e-17
mono_stripe	string dataset	"Ru/C"
geometry	group	

`type`
Multilayer type.

`energy`
Peak of the spectrum that the monochromator selects. Since units is not defined this field is in J and corresponds to 10 keV.

`energy_error`
Standard deviation of the spectrum that the monochromator selects. Since units is not defined this field is in J.

`mono_stripe`
Type of multilayer coating or crystal.

4.3.2.5 Detector

This class holds information about the detector used during the experiment. If more than one detector are used they will be all listed as detector_N.

Table 16: Detector Group Members

Member	Type	Example
manufacturer	string dataset	"CooKe Corporation"
model	string dataset	"pco dimax"
serial_number	string dataset	"1234XW2"
geometry	group	
output_data	string dataset	"/exchange"

`manufacturer`
The detector manufacturer.

`model`
: The detector model.

`serial_number`
: The detector serial number .

`output_data`
: String HDF5 path to the exchange group where the detector output data is located.

4.4 Provenance

Data provenance is the documentation of all transformations, analyses and interpretations of data performed by a sequence of process functions or `actors`.

Maintaining this history allows for reproducible data. The Data Exchange format tracks provenance by allowing each actor to append provenance information to a process table. The provenance process table tracks the execution order of a series of processes by appending sequential entries in the process table.

Scientific users will not generally be expected to maintain data in this group. The expectation is that analysis pipeline tools will automatically record process steps using this group. In addition, it is possible to re-run an analysis using the information provided here.

Table 17: Process table to log actors activity

actor	start_time	end_time	status	message	reference	description
gridftp	21:15:22	21:15:23	FAILED	auth. error	/provenance/griftp	transfer detector to cluster
gridftp	21:15:26	21:15:27	FAILED	auth. error	/provenance/griftp	transfer detector to cluster
gridftp	21:17:28	22:15:22	SUCCESS	OK	/provenance/griftp	transfer detector to cluster
norm	22:15:23	22:30:22	SUCCESS	OK	/provenance/norm	normalize the raw data
rec	22:30:23	22:50:22	SUCCESS	OK	/provenance/rec	reconstruct the norm. data
convert	22:50:23		RUNNING	OK	/provenance/export	convert reconstructed data
gridftp			QUEUED		/provenance/griftp_2	transfer data to user

`actor`
: Name of the process in the pipeline stage that is executed at this step.

start_time
: Time the process started.

end_time
: TIme the process ended.

status
: Current process status. May be one of the following: QUEUED, RUNNING, FAILED, or SUCCESS.

message
: A process specific message generated by the process. It may be a confirmation that the process was successful, or a detailed error message, for example.

reference
: Path to a process description group. The process description group contains all metadata to perform the specific process. This reference is simply the HDF5 path within this file of the technique specific process description

group. The process description group should contain all parameters necessary to run the process, including the name and version of any external analysis tool used to process the data. It should also contain input and output references that point to the exchange_N groups that contain the input and output datasets of the process.

description
Process description.

5 Data Exchange for X-ray Tomography

This section describes extensions and additions to the core Data Exchange format for X-ray Tomography. We begin with the extensions to the exchange and instrument groups, and then describe the tomography process groups.

5.1 Exchange Group for X-ray Tomography

In x-ray tomography, the 3D arrays representing the most basic version of the data include projections, dark, and white fields. It is *mandatory* that there is at least one dataset named data in each exchange group. Most data analysis and plotting programs will primarily focus in this group.

Table 18: Exchange Group Members for Tomography

Member	Type	Example/Attributes
`title`	string dataset	"raw absorption tomo"
data	3D dataset	axes: "theta:y:x"
`x`	dimension scale 2	
`y`	dimension scale 1	
`theta`	dimension scale 0	units: "deg"
`data_dark`	3D dataset	axes: "theta_dark:y:x"
`theta_dark`	dimension scale 0	units: "deg"
`data_white`	3D dataset	axes: "theta_white:y:x"
`theta_white`	dimension scale 0	units: "deg"

`title`

This is the data title.

data

A tomographic data set consists of a series of projections (data), dark field (data_dark), and white field (data_white) images. The dark and white fields must have the same projection image dimensions and can be collected at any time before, after or during the projection data collection. The angular position of the tomographic rotation axis, theta, can be used to keep track of when the dark and white images are collected. These datasets are saved in 3D arrays using, by default, the natural HDF5 order of a multidimensional array (rotation axis, ccd y, ccd x), i.e. with the fastest changing dimension being the last dimension, and the slowest changing dimension being the first dimension. If using the default dimension order, the axes attribute "theta:y:x" can be omitted. The `axes` attribute is mandatory if the 3D arrays use a different axes order. This could be the case when, for example, the arrays are optimized for sinogram read (`axes` = "y:theta:x"). As no units are specified the data is assumed to be in "counts" with the axes (x, y) in pixels.

`data_dark, data_white`
: The dark field and white fields must have the same dimensions as the projection images and can be collected at any time before, during, or after the projection data collection. To specify where dark and white images were taken, specify the axes attribute with "theta_dark:y:x" and "theta_white:y:x" and provide theta_dark and theta_white vector datasets that specify the rotation angles where they were collected.

`x, y`
: X and y are vectors storing the dimension scale for the second and third data array dimension. If x, y are not defined, the second and third dimensions of the data array are assumed to be in pixels.

`theta, theta_dark, theta_white`
: Theta is a vector dataset storing the projection angular positions. If theta is not defined the projections are assumed to be collected at equally spaced angular interval between 0 and 180 degree. The dark field and white fields can be collected at any time before, during, or after the projection data. Theta_dark, and theta_white store the position of the tomographic rotation axis when the corresponding dark and white images are collected. If theta_dark and theta_white are missing the corresponding data_dark and data_white are assumed to be collected all at the beginning or at the end of the projection data collection.

5.2 Instrument Group for Tomography

The instrument group for X-ray tomography introduces an extended detector group definition along with new interferometer group. The extended instrument group is as shown in Table **??**.

Table 19: Instrument Group for Tomography

Member	Type	Example
`name`	string dataset	"XSD/2-BM"
source	group	same as core
shutter_*N*	group	same as core
attenuator_*N*	group	same as core
monochromator	group	same as core
interferometer	group	new
setup	group	new
acquisition	group	new
detector_*N*	group	extended from core

5.2.1 Interferometer

This group stores the interferometer parameters.

Table 20: Interferometer Group Members

Member	Type	Example
`start_angle`	float dataset	0.000
`grid_start`	float dataset	0.000
`grid_end`	float dataset	2.4e-6
`grid_position_for_scan`	float dataset	1.3e-6
`number_of_grid_steps`	int dataset	8
geometry	group	

`start_angle`
: Interferometer start angle.

`grid_start`
: Interferometer grid start angle.

`grid_end`
: Interferometer grid end angle.

`grid_position_for_scan`
: Interferometer grid position for scan.

`number_of_grid_steps`
: Number of grid steps.

5.2.2 Setup and Acquisition Parameters

Logging instrument setup parameters (for static setup) and storing acquisition configuration parameters (for the scan engine setup) is not defined by Data Exchange because is specific and different for each instrument. For these information Data Exchange defines a setup and acquisition group under the instrument group and leaves each facility free to store what is relevant for a specific instrument.

5.2.3 Detector Group for Tomography

This class holds information about the detector used during the experiment. If more than one detector are used they will be all listed as detector_N. In full field imaging the detector consists of a CCD camera, microscope objective and a scintillator screen. Raw data recorded by a detector as well as its position and geometry should be stored in this class.

`manufacturer`
: The detector manufacturer.

`model`
: The detector model.

Table 21: Detector Group Members for Tomography

Member	Type	Example
manufacturer	string dataset	"CooKe Corporation"
model	string dataset	"pco dimax"
serial_number	string dataset	"1234XW2"
bit_depth	integer	12
x_pixel_size	float	6.7e-6
y_pixel_size	float	6.7e-6
x_dimension	integer	2048
y_dimension	integer	2048
x_binning	integer	1
y_binning	integer	1
operating_temperature	float	270
exposure_time	float	1.7e-3
frame_rate	integer	2
output_data	string dataset	"/exchange"
roi	group	
objective_N	group	
scintillator	group	
counts_per_joule	float	unitless
basis_vectors	float array	length
corner_position	3 floats	length
geometry	group	

`serial_number`
: The detector serial number .

`bit_depth`
: The detector bit depth.

`x_pixel_size, y_pixel_size`
: Physical detector pixel size (m).

`x_dimension, y_dimension`
: The detector horiz./vertical dimension.

`x_binning, y_binning`
: If the data are collected binning the detector x_binning and y_binning store the binning factor.

`operating_temperature`
: The detector operating temperature (K).

`exposure_time`
: The detector exposure time (s).

`frame_rate`
: The detector frame rate (fps). This parameter is set for fly scan

`roi`
: The detector selected Region Of Interest (ROI).

`objective_N`
: List of the visible light objectives mounted between the detector and the scintillator screen.

`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:

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

`basis_vectors`
: A matrix with the basis vectors of the detector data.

`corner_position`
: The x, y and z coordinates of the corner of the first data element.

`geometry`
: 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.

5.2.3.1 ROI Group describing the region of interest (ROI) of the image actually collected, if smaller than the full CCD.

Table 22: ROI Group Members

Member	Type	Example
`name`	string dataset	"center third"
`x1`	integer	256
`y1`	integer	256
`x2`	integer	1792
`y2`	integer	1792

`x1`
: Left pixel position.

`y1`
: Top pixel position.

`x2`
: Right pixel position.

`y2`
: Bottom pixel position.

5.2.3.2 Objective

Group describing the microscope objective lenses used.

Table 23: Objective Group Members

Member	Type	Example
manufacturer	string dataset	"Zeiss"
model	string dataset	"Axioplan"
magnification	float dataset	5
numerical_aperture	float dataset	0.8
geometry	group	

`manufacturer`
: Lens manufacturer.

`model`
: Lens model.

`magnification`
: Lens specified magnification.

`numerical_aperture`
: The numerical aperture (N.A.) is a measure of the light-gathering characteristics of the lens.

5.2.3.3 Scintillator

Group describing the visible light scintillator coupled to the CCD camera objective lens.

Table 24: Scintillator Group Members

Member	Type	Example
manufacturer	string dataset	"Crytur"
serial_number	string dataset	"12"
name	string dataset	"Yag polished"
type	string dataset	"Yag on Yag"
scintillating_thickness	float dataset	5e-6
substrate_thickness	float dataset	1e-4
geometry	group	

`manufacturer`
: Scintillator Manufacturer.

`serial_number`
: Scintillator serial number.

`name`
: Scintillator name.

`scintillating_thickness`
Scintillator thickness.

`substrate_thickness`
Scintillator substrate thickness.

5.2.3.4 Geometry

This class holds the position and orientation of a component for tomography.

Table 25: Geometry Group Members

Member	Type	Example
translation	group	
orientation	group	

translation
The position of the object with respect to the origin of your coordinate system.

orientation
The rotation of the object with respect to your coordinate system.

5.2.3.4.1 Translation

This is the description for the general spatial location of a component for tomography.

Table 26: Translation Group Members

Member	Type	Example
distances	3 float array dataset	(0, 0.001, 0)

`distances`
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 when the sample is first aligned in the beam). If distances does not have the attribute units set then the units are in meters.

5.2.3.4.2 Orientation

This is the description for the orientation of a component for tomography.

Table 27: Orientation Group Members

Member	Type	Example
value	6 float array dataset	

`value`
: Dot products between the local and the global unit vectors. Unitless

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. The third row can be recovered by using the fact that the basis vectors are orthonormal.

5.3 Tomography Process Descriptions

This section documents a set of process descriptions for tomography data movement and processing. These process description groups are used in a data processing pipeline - each group provides the metadata for one stage in the pipeline.

5.3.1 Sinogram Process Description

The sinogram process description group contains metadata required to generate sinograms from projection data. The input data is a projection ordered data cube, and the output is a sinogram ordered data cube. The output is stored in a new exchange group.

Table 28: Sinogram Group Members

Member	Type	Example
`name`	string dataset	
`version`	string dataset	1.0
`input_data`	string dataset	"/exchange_1"
`output_data`	string dataset	"/exchange_2"

`name`
: Algorithm name.

`version`
: Algorithm version.

`input_data`
: Reference to the exchange group containing the projection ordered input data.

`output_data`
: Reference to the exchange group that will contain the sinogram ordered output data.

5.3.2 Ring Removal Process Description

The ring removal process description group contains information required to run a ring_removal processing step.

Table 29: Ring Removal Group Members

Member	Type	Example
`name`	string dataset	
`version`	string dataset	1.0
`input_data`	string dataset	"/exchange_2"
`output_data`	string dataset	"/exchange_3"
`coefficient`	float dataset	1.0

`name`
: Algorithm name.

`version`
: Algorithm version.

`input_data`
: Reference to the exchange group containing input data.

`output_data`
: Path to the exchange group containing output data.

`coefficient`

5.3.3 Reconstruction Process Description

The Reconstruction process description group contains metadata required to run a tomography reconstruction. The specific algorithm is described in a separate group.

`name`
: Reconstruction tool name.

`version`
: Tool version.

`input_data`
: Reference to the exchange group containing input data.

Table 30: Reconstruction Group Members.

Member	Type	Example
name	string dataset	
version	string dataset	1.0
input_data	string dataset	"/exchange_3"
output_data	string dataset	"/exchange_4"
reconstruction_slice_start	int dataset	1000
reconstruction_slice_end	int dataset	1030
rotation_center	float dataset	1048.50
algorithm	group	

`output_data`
Reference to the exchange group containing output data.

`reconstruction_slice_start`
First reconstruction slice.

`reconstruction_slice_end`
Last reconstruction slice.

`rotation_center`
Center of rotation in pixels.

`algorithm`
Algorithm group describing reconstruction algorithm parameters.

5.3.3.1 Algorithm

The Algorithm group contains information required to run a tomography reconstruction algorithm.

`name`
Reconstruction method name: SART, EM, FBP, GridRec.

`version`
Algorithm version.

`implementation`
CPU or GPU.

`number_of_nodes`
Number of nodes to use on cluster. This parameter is set when the reconstruction is parallelized and run on a cluster.

`type`
Tomography reconstruction method: analytic or iterative.

`iterative_stop_condition`
iteration_max, projection_threshold, difference_threshold_percent, difference_threshold_value.

Table 31: Algorithm Group Members

Member	Type	Example
name	string dataset	"SART"
version	string dataset	"1.0"
implementation	string dataset	"GPU"
number_of_nodes	int dataset	16
type	string dataset	"Iterative"
iterative_stop_condition	string dataset	"iteration_max"
iterative_iteration_max	int dataset	200
iterative_projection_threshold	float dataset	
iterative_difference_threshold_percent	float dataset	
iterative_difference_threshold_value	float dataset	
iterative_regularization_type	string dataset	"total_variation"
iterative_regularization_parameter	float dataset	
iterative_step_size	float dataset	0.3
iterative_sampling_step_size	float dataset	0.2
analytic_filter	string dataset	"Parzen"
analytic_padding	float dataset	0.50
analytic_processed_periods	float dataset	1
analytic_processed_number_of_steps	int dataset	7

`iterative_iteration_max`
: Maximum number of iterations.

`iterative_projection_threshold`
: The threshold of projection difference to stop the iterations as p in $|y - Ax_n| < p$.

`iterative_difference_threshold_percent`
: The threshold of reconstruction difference to stop the iterations as p in $|x_{n+1}|/|x_n| < p$.

`iterative_difference_threshold_value`
: The threshold of reconstruction difference to stop the iterations as p in $|x_{n+1}| - |x_n| < p$.

`iterative_regularization_type`
: total_variation, none.

`iterative_regularization_parameter`
: lambda/alpha value in $(y - A_x)^2 + \alpha * L_1(x)$.

`iterative_step_size`
: Step size between iterations in iterative methods as δ_t in $x_{n+1} = x_n + \delta_t * f(x_n)$.

`iterative_sampling_step_size`
: Step size used for forward projection calculation in iterative methods.

`analytic_filter`
: Filter type.

`analytic_padding`

`analytic_processed_periods`
: number of processed periods of the collected phase stepping curve (differential phase contrast - grating).

`analytic_processed_number_of_steps`
: total number of processed phase steps (differential phase contrast - grating).

5.3.4 Gridftp Process Description

The gridftp process description group contains metadata required to transfer data between two gridftp endpoints. This assumes a third party transfer.

Table 32: Gridftp Group Members

Member	Type	Example
`name`	string dataset	
`version`	string dataset	1.0
`source_URL`	string dataset	"gsiftp://host1/path"
`dest_URL`	string dataset	"gsiftp://host2/path"

`name`
: GridFTP tool name.

`version`
: Tool version.

`source_URL`
: A gsiftp URL for the source of the transfer.

`dest_URL`
: A gsiftp URL for the destination of the transfer.

5.3.5 Export Process Description

The export process description group contains metadata required to extract and convert data from a Data Exchange (HDF5) file into another format.

`name`
: Export tool name.

Table 33: Export Group Members

Member	Type	Example
`name`	string dataset	
`version`	string dataset	"1.0"
`input_data`	string dataset	"/exchange_4"
`output_URL`	string dataset	"file://host/path"
`output_data_format`	string dataset	"TIFF"
`output_data_scaling_max`	float dataset	0.005
`output_data_scaling_min`	float dataset	-0.00088

`version`
: Tool version.

`input_data`
: Reference to the exchange group containing the input data.

`output_URL`
: A file path and name, either plain or in URL format: file://host/path/file.tif

`output_data_format`

`output_data_scaling_max`

`output_data_scaling_min`

6 Data Exchange for X-ray Photon Correlation

Extensions and additions to Data Exchange for X-ray Photon Correlation are being defined and will be incorporated into this reference soon.

7 Data Exchange for X-ray Fluorescence

Extensions and additions to Data Exchange for X-ray Fluorescence are being defined and will be incorporated into this reference soon.

8 Code Examples

Below is the Python code to generate the Data Exchange files decribed in the diagrams in Fig. **??**, **??**, **??** (right), **??** (right), **??** (left) as well as full implementation for tomography.

All the code examples as well as the resulting Data Exchange files are available from http://www.aps.anl.gov/DataExchange/.

8.1 Creating Data Exchange files using Pyhton

Listing 1: Python Code for Fig. **??**

```
#=========================================================================
# Sample Python code to write Data Exchange Format
#
# Date: 2011-09-01
# Updated: 2013-04-21
#=========================================================================

import h5py
import numpy as np

def write_example(filename):

    # --- prepare data ---

    # Generate fake data
    rawdata = np.ones(180 * 256 * 256, np.uint16).reshape(180, 256, 256)

    # x, y and z ranges
    x = np.arange(256)
    y = np.arange(256)
    z = np.arange(180);

    # --- create file ---

    # Open HDF5 file
    f = h5py.File(filename, 'w')

    # Create basic definitions in root
    ds = f.create_dataset('implements', data = "exchange")

    # --- exchange definition ---

    # Exchange HDF5 group
    # /exchange
    exchangeGrp = f.create_group("exchange")

    # Create core HDF5 dataset in exchange group for 180 deep stack
    # of x,y images /exchange/data
    ds = exchangeGrp.create_dataset('data', data = rawdata)
    ds.attrs['units'] = "counts"
```

```
    # --- All done ---
    f.close()

if __name__ == '__main__':

    write_example('/tmp/python/DataExchange-example0.h5')
#=====================================================================
#
#=====================================================================
```

Listing 2: Python Code for Fig. **??**

```
#=====================================================================
# Sample Python code to write Data Exchange Format
#
# Date: 2013-04-21
#
#=====================================================================

import h5py
import numpy as np

def write_example(filename):

    # --- prepare data ---

    # Generate fake data
    rawdata = np.ones(180 * 256 * 256, np.uint16).reshape(180, 256, 256)
    rawdata_white = np.ones(2 * 256 * 256, np.uint16).reshape(2, 256, 256)
    rawdata_dark = np.zeros(10 * 256 * 256, np.uint16).reshape(10, 256, 256)

    # x, y and z ranges
    x = np.arange(256)
    y = np.arange(256)
    z = np.arange(180);

    # --- create file ---

    # Open HDF5 file
    f = h5py.File(filename, 'w')

    # Create basic definitions in root
    ds = f.create_dataset('implements', data = "exchange")

    # --- exchange definition ---

    # Exchange HDF5 group
    # /exchange
    exchangeGrp = f.create_group("exchange")

    # Create core HDF5 dataset in exchange group for 180 deep stack
    # of x,y images /exchange/data
    ds = exchangeGrp.create_dataset('data', data = rawdata)
    ds.attrs['units'] = "counts"
```

```
    # Create HDF5 dataset in exchange group for dark data
    # /exchange/data_dark
    ds = exchangeGrp.create_dataset('data_dark', data = rawdata_dark)
    ds.attrs['units'] = "counts"

    # Create HDF5 dataset in exchange group for white data
    # /exchange/data_white
    ds = exchangeGrp.create_dataset('data_white', data = rawdata_white)
    ds.attrs['units'] = "counts"

    # --- All done ---
    f.close()

if __name__ == '__main__':

    write_example('/tmp/python/DataExchange-example1.h5')
#=====================================================================
#
#=====================================================================
```

Listing 3: Python Code for Fig. **??** (right)

```
#=====================================================================
# Sample Python code to write Data Exchange Format
#
# Date: 2013-04-21
#
#=====================================================================

import h5py
import numpy as np

def write_example(filename):

    # --- prepare data ---

    # Generate fake data
    rawdata = np.ones(180 * 256 * 256, np.uint16).reshape(180, 256, 256)
    rawdata_white = np.ones(2 * 256 * 256, np.uint16).reshape(2, 256, 256)
    rawdata_dark = np.zeros(10 * 256 * 256, np.uint16).reshape(10, 256, 256)

    # x, y and z ranges
    x = np.arange(256)
    y = np.arange(256)
    z = np.arange(180);

    # Fabricated theta values
    theta = (z / float(180)) * 180.0
    theta_white = (0.0, 180.0)
    theta_dark = (0.0, 0.0, 0.0, 0.0, 0.0, 180.0, 180.0, 180.0, 180.0, 180.0)

    # --- create file ---

    # Open HDF5 file
    f = h5py.File(filename, 'w')

    # Create basic definitions in root
    ds = f.create_dataset('implements', data = "exchange")

    # --- exchange definition ---

    # Exchange HDF5 group
    # /exchange
    exchangeGrp = f.create_group("exchange")
```

```
    # Create core HDF5 dataset in exchange group for 180 deep stack
    # of x,y images /exchange/data
    ds = exchangeGrp.create_dataset('data', data = rawdata)
    ds.attrs['units'] = "counts"
    ds.attrs['axes'] = "theta:y:x"

    # Create HDF5 dataset in exchange group for dark data
    # /exchange/data_dark
    ds = exchangeGrp.create_dataset('data_dark', data = rawdata_dark)
    ds.attrs['units'] = "counts"
    ds.attrs['axes'] = "theta_dark:y:x"

    # Create HDF5 dataset in exchange group for white data
    # /exchange/data_white
    ds = exchangeGrp.create_dataset('data_white', data = rawdata_white)
    ds.attrs['units'] = "counts"
    ds.attrs['axes'] = "theta_white:y:x"

    # Create HDF5 dataset in exchange group for theta
    # /exchange/theta
    ds = exchangeGrp.create_dataset('theta', data = theta)
    ds.attrs['units'] = "degrees"

    # Create HDF5 dataset in exchange group for theta_dark
    # /exchange/theta_dark
    ds = exchangeGrp.create_dataset('theta_dark', data = theta_dark)
    ds.attrs['units'] = "degrees"

    # Create HDF5 dataset in exchange group for theta_white
    # /exchange/theta_white
    ds = exchangeGrp.create_dataset('theta_white', data = theta_white)
    ds.attrs['units'] = "degrees"

    # --- All done ---
    f.close()

if __name__ == '__main__':
    write_example('/tmp/python/DataExchange-example2.h5')
#=====================================================================
#
#=====================================================================
```

Listing 4: Python Code for Fig. **??** (right)

```
#=====================================================================
# Sample Python code to write Data Exchange Format
#
# Date: 2013-04-21
#
#=====================================================================

import h5py
import numpy as np

def write_example(filename):

    # --- prepare data ---

    # Generate fake raw data
    rawdata = np.ones(180 * 256 * 256, np.uint16).reshape(180, 256, 256)

    # x, y and z ranges
    x = np.arange(128)
    y = np.arange(128)
    z = np.arange(180);

    # --- create file ---

    # Open HDF5 file
    f = h5py.File(filename, 'w')

    # Create basic definitions in root
    ds = f.create_dataset('implements', data = "measurement:exchange")

    # --- exchange definition ---

    # Exchange HDF5 group
    # /exchange
    exchangeGrp = f.create_group("exchange")

    # Create core HDF5 dataset in exchange group for 180 deep stack of x,y
    # images /exchange/data
    ds = exchangeGrp.create_dataset('data', data = rawdata, \
                                    compression='gzip', compression_opts=4)

    ds.attrs['description'] = "Projection data"
```

```
    ds.attrs['units'] = "counts"

    # Create HDF5 group measurement
    # /measurement
    measurementGrp = f.create_group("measurement")

    # Create HDF5 subgroup
    # /measurement/sample
    sampleGrp = measurementGrp.create_group("sample")
    sads1 = sampleGrp.create_dataset('name', data = "Minivirus")
    sads8 = sampleGrp.create_dataset('temperature', data = 200.0, dtype='d')
    sads8.attrs['units'] = "celsius"

    # --- All done ---
    f.close()

if __name__ == '__main__':

    write_example('/tmp/python/DataExchange-example3.h5')
#==========================================================================
#
#==========================================================================
```

Listing 5: Python Code for Fig. **??** (right)

```
#=====================================================================
# Sample Python code to write Data Exchange Format
#
# Date: 2013-04-21
#
#=====================================================================

import h5py
import numpy as np

def write_example(filename):

    # --- prepare data ---

    # Generate fake raw data
    rawdata = np.ones(180 * 256 * 256, np.uint16).reshape(180, 256, 256)

    # x, y and z ranges
    x = np.arange(128)
    y = np.arange(128)
    z = np.arange(180);

    # --- create file ---

    # Open HDF5 file
    f = h5py.File(filename, 'w')

    # Create basic definitions in root
    ds = f.create_dataset('implements', data = "measurement:exchange")

    # --- exchange definition ---

    # Exchange HDF5 group
    # /exchange
    exchangeGrp = f.create_group("exchange")

    # Create core HDF5 dataset in exchange group for 180 deep stack of x,y
    # images /exchange/data
    ds = exchangeGrp.create_dataset('data', data = rawdata, \
                                    compression='gzip', compression_opts=4)

    ds.attrs['description'] = "Projection data"
```

```
    ds.attrs['units'] = "counts"

    # Create HDF5 group measurement
    # /measurement
    measurementGrp = f.create_group("measurement")

    # Create HDF5 subgroup
    # /measurement/sample
    sampleGrp = measurementGrp.create_group("sample")
    sads1 = sampleGrp.create_dataset('name', data = "Minivirus")
    sads8 = sampleGrp.create_dataset('temperature', data = 100.0, dtype='d')
    sads8.attrs['units'] = "celsius"

    # Create HDF5 subgroup
    # /measurement/instrument
    instrumentGrp = measurementGrp.create_group("instrument")
    ids1 = instrumentGrp.create_dataset('name', data = "APS 2-BM")

    # Create HDF5 subgroup
    # /measurement/instrument/monochromator
    monochromatorGrp = instrumentGrp.create_group("monochromator")
    mds1 = monochromatorGrp.create_dataset('name', data = "DMM")
    mds2 = monochromatorGrp.create_dataset('energy', data = 10.00, dtype='d')
    mds2.attrs['units'] = "keV"

    # --- All done ---
    f.close()

if __name__ == '__main__':

    write_example('/tmp/python/DataExchange-example4.h5')
#=====================================================================
#
#=====================================================================
```

Listing 6: Python Code for Data Exchange full implementation for tomography

```
#=====================================================================
# Sample Python code to write Data Exchange Format
#
# Date: 2013-04-21
#
#=====================================================================

import h5py
import numpy as np

def write_example(filename):

    # --- prepare data ---

    # Generate fake raw data
    rawdata = np.ones(180 * 256 * 256, np.uint16).reshape(180, 256, 256)
    rawdata_white = np.ones(2 * 256 * 256, np.uint16).reshape(2, 256, 256)
    rawdata_dark = np.zeros(10 * 256 * 256, np.uint16).reshape(10, 256, 256)

    # Generate fake normalized data
    normalizeddata = np.ones(180 * 256 * 256, \
                             np.float64).reshape(180, 256, 256)

    # Generate fake reconstructed data
    reconstructeddata = np.ones(256 * 256 * 256, \
                                np.float64).reshape(256, 256, 256)

    # x, y and z ranges
    x = np.arange(128)
    y = np.arange(128)
    z = np.arange(180);

    # Fabricated theta values
    theta = (z / float(180)) * 180.0
    theta_white = (0.0, 180.0)
    theta_dark = (0.0, 0.0, 0.0, 0.0, 0.0, 180.0, 180.0, 180.0, 180.0, 180.0)

    # --- create file ---

    # Open HDF5 file
    f = h5py.File(filename, 'w')
```

```
# Create basic definitions in root
ds = f.create_dataset('implements', \
                data = "exchange:exchange_2:exchange_3:measurement")

# --- exchange definition ---

# Exchange HDF5 group
# /exchange
exchangeGrp = f.create_group("exchange")

# Create core HDF5 dataset in exchange group for 180 deep stack
# of x,y images /exchange/data
ds = exchangeGrp.create_dataset('data', data = rawdata, \
                                compression='gzip', compression_opts=4)

ds.attrs['description'] = "transmission"
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta:y:x"
ds1 = exchangeGrp.create_dataset('title', \
                                 data = "tomography raw projections")

# Create HDF5 dataset in exchange group for dark data
# /exchange/data_dark
ds = exchangeGrp.create_dataset('data_dark', \
                                data = rawdata_dark, \
                                compression='gzip', compression_opts=4)
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta_dark:y:x"

# Create HDF5 dataset in exchange group for white data
# /exchange/data_white
ds = exchangeGrp.create_dataset('data_white', \
                                data = rawdata_white, \
                                compression='gzip', compression_opts=4)
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta_white:y:x"

# Create HDF5 dataset in exchange group for theta
# /exchange/theta
ds = exchangeGrp.create_dataset('theta', data = theta)
ds.attrs['units'] = "degrees"

# Create HDF5 dataset in exchange group for theta_dark
```

```
# /exchange/theta_dark
ds = exchangeGrp.create_dataset('theta_dark', data = theta_dark)
ds.attrs['units'] = "degrees"

# Create HDF5 dataset in exchange group for theta_white
# /exchange/theta_white
ds = exchangeGrp.create_dataset('theta_white', data = theta_white)
ds.attrs['units'] = "degrees"

# Exchange HDF5 group
# /exchange_2
# this will be the out_put of the normalization process
exchange2Grp = f.create_group("exchange_2")
ds2 = exchange2Grp.create_dataset('title', \
                        data = "tomography normalized projections")

# Create core HDF5 dataset in exchange group for 180 deep stack
# of x,y images /exchange_2/data
ds = exchange2Grp.create_dataset('data', data = normalizeddata, \
                                 compression='gzip', \
                                 compression_opts=4)
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta:y:x"
# Create HDF5 dataset in exchange_2 group for theta
# /exchange_2/theta
ds = exchange2Grp.create_dataset('theta', data = theta)
ds.attrs['units'] = "degrees"

# Exchange HDF5 group
# /exchange_3
# this will be the out_put of the reconstruction process
exchange3Grp = f.create_group("exchange_3")
ds3 = exchange3Grp.create_dataset('title', \
                                  data = "tomography reconstructions")

# Create core HDF5 dataset in exchange group for 180 deep stack of x,y
# images /exchange_3/data
ds = exchange3Grp.create_dataset('data', \
                                 data = reconstructeddata, \
                                 compression='gzip', \
                                 compression_opts=4)
ds.attrs['units'] = "density"
ds.attrs['axes'] = "z:y:x"
```

```
# Create HDF5 group measurement
# /measuremen
measurementGrp = f.create_group("measurement")

# Create HDF5 subgroup
# /measurement/instrument
instrumentGrp = measurementGrp.create_group("instrument")

# Create HDF5 subgroup
# /measurement/instrument/source
sourceGrp = instrumentGrp.create_group("source")
sods1 = sourceGrp.create_dataset('name', data = "APS")
sods2 = sourceGrp.create_dataset('date_time', data = "2012-07-31T21:15:23+

sods3 = sourceGrp.create_dataset('beamline', data = "2-BM")
sods4 = sourceGrp.create_dataset('current', data = 101.199, dtype='d')
sods4.attrs['units'] = "mA"
sods5 = sourceGrp.create_dataset('energy', data = 7.0, dtype='d')
sods5.attrs['units'] = "GeV"
sods6 = sourceGrp.create_dataset('mode', data = "TOPUP")

# Create HDF5 subgroup
# /measurement/instrument/attenuator
attenuatorGrp = instrumentGrp.create_group("attenuator")
ats1 = attenuatorGrp.create_dataset('thickness', data = 1e-3, \
                                    dtype='d')
ats2 = attenuatorGrp.create_dataset('type', data = "Al")

# Create HDF5 subgroup
# /measurement/instrument/monochromator
monochromatorGrp = instrumentGrp.create_group("monochromator")
mds1 = monochromatorGrp.create_dataset('type', data = "Multilayer")
mds2 = monochromatorGrp.create_dataset('energy', data = 19.26, \
                                       dtype='d')
mds2.attrs['units'] = "keV"
mds3 = monochromatorGrp.create_dataset('energy_error', data = 1e-3, \
                                       dtype='d')
mds3.attrs['units'] = "keV"
mds4 = monochromatorGrp.create_dataset('mono_stripe', data = "Ru/C")

# Create HDF5 subgroup
# /measurement/instrument/detector
```

```
detectorGrp = instrumentGrp.create_group("detector")

# define the detector subgroup members of instrument
dds1 = detectorGrp.create_dataset('manufacturer', \
                                  data = "CooKe Corporation")
dds2 = detectorGrp.create_dataset('model', data = "pco dimax")
dds3 = detectorGrp.create_dataset('serial_number', data = "1234XW2")
dds4 = detectorGrp.create_dataset('bit_depth', data = 12, dtype='i')

# for x_pixel_size and y_pixel_size if the attributes units is not
# specified then these are in meters
dds5 = detectorGrp.create_dataset('x_pixel_size', data = 6.7e-6, \
                                  dtype='f')
dds6 = detectorGrp.create_dataset('y_pixel_size', data = 6.7e-6, \
                                  dtype='f')
dds7 = detectorGrp.create_dataset('x_dimentions', data = 2048, \
                                  dtype='i')
dds8 = detectorGrp.create_dataset('y_dimentions', data = 2048, \
                                  dtype='i')
dds9 = detectorGrp.create_dataset('x_binning', data = 1, dtype='i')
dds10 = detectorGrp.create_dataset('y_binning', data = 1, dtype='i')

# for operating_temperature if the attributes units is not
# specified then this is in K
dds11 = detectorGrp.create_dataset('operating_temperature', \
                                   data = 270, dtype='f')

# for exposure_time the attributes units is specified as ms
dds12 = detectorGrp.create_dataset('exposure_time', data = 170, \
                                   dtype='d')
dds12.attrs['units'] = "ms"

dds13 = detectorGrp.create_dataset('frame_rate', data = 3, dtype='i')
dds14 = detectorGrp.create_dataset('output_data', data = "/exchange")

roiGrp = detectorGrp.create_group("roi")
rds1 = roiGrp.create_dataset('name', data = "center third")
rds2 = roiGrp.create_dataset('x1', data = 256, dtype='i')
rds3 = roiGrp.create_dataset('y1', data = 256, dtype='i')
rds4 = roiGrp.create_dataset('x2', data = 1792, dtype='i')
rds5 = roiGrp.create_dataset('y2', data = 1792, dtype='i')

objectiveGrp = detectorGrp.create_group("objective")
```

```
ods1 = objectiveGrp.create_dataset('manufacturer', data = "Zeiss")
ods2 = objectiveGrp.create_dataset('model', \
                                   data = "Plan-NEOFLUAR 1004-072")
ods3 = objectiveGrp.create_dataset('magnification', \
                                   data = 20, dtype='d')
ods4 = objectiveGrp.create_dataset('numerical_aperture', \
                                   data = 0.5, dtype='d')


scintillatorGrp = detectorGrp.create_group("scintillator")
sds1 = scintillatorGrp.create_dataset('manufacturer', data = "Crytur")
sds2 = scintillatorGrp.create_dataset('serial_number', data = "12")
sds3 = scintillatorGrp.create_dataset('name', data = "YAG polished")
sds4 = scintillatorGrp.create_dataset('type', data = "YAG on YAG")
sds5 = scintillatorGrp.create_dataset('scintillating_thickness', \
                                      data = 5e-6, dtype='d')
sds6 = scintillatorGrp.create_dataset('substrate_thickness', \
                                      data = 1e-4, dtype='d')


# Create HDF5 subgroup
# /measurement/sample
sampleGrp = measurementGrp.create_group("sample")
sads1 = sampleGrp.create_dataset('name', data = "Hornby_b")
sads2 = sampleGrp.create_dataset('description', data = "test sample")
sads3 = sampleGrp.create_dataset('preparation_date', \
                                 data = "2011-07-31T21:15:23+0600")
sads4 = sampleGrp.create_dataset('chemical_formula', data = "unknown")
sads5 = sampleGrp.create_dataset('mass', data = 0.25, dtype='d')
sads5.attrs['units'] = "g"
sads7 = sampleGrp.create_dataset('enviroment', data = "air")
sads8 = sampleGrp.create_dataset('temperature', data = 120.0, \
                                 dtype='d')
sads8.attrs['units'] = "Celsius"
sads9 = sampleGrp.create_dataset('temperature_set', data = 130.0, \
                                 dtype='d')
sads9.attrs['units'] = "Celsius"

# Create HDF5 subgroup
# /measurement/sample/geometry
geometryGrp = sampleGrp.create_group("geometry")

# Create HDF5 subgroup
```

```
    # /measurement/sample/geometry/translation
    translationGrp = geometryGrp.create_group("translation")
    trds1 = translationGrp.create_dataset('distances', \
                                          data = [0, 0, 0], dtype='d')
    trds1.attrs['axes'] = "z:y:x"
    trds1.attrs['units'] = "m"

    # Create HDF5 subgroup
    # /measurement/sample/experimenter
    experimenterGrp = sampleGrp.create_group("experimenter")
    exrds1 = experimenterGrp.create_dataset('name', data = "John Doe")
    exrds2 = experimenterGrp.create_dataset('role', data = "Project PI")
    exrds3 = experimenterGrp.create_dataset('affiliation', \
                        data = "University of California, Berkeley")
    exrds4 = experimenterGrp.create_dataset('address', \
                        data = "EPS UC Berkeley CA 94720 4767 USA")
    exrds5 = experimenterGrp.create_dataset('phone', \
                        data = "+1 123 456 0000")
    exrds6 = experimenterGrp.create_dataset('e-mail', \
                        data = "johndoe@berkeley.edu")
    exrds7 = experimenterGrp.create_dataset('facility_user_id', \
                        data = "a123456")

    # Create HDF5 subgroup
    # /measurement/sample/experiment
    experimentGrp = sampleGrp.create_group("experiment")
    exds1 = experimentGrp.create_dataset('proposal', data = "1234")
    exds2 = experimentGrp.create_dataset('activity', data = "e11218")
    exds3 = experimentGrp.create_dataset('safety', data = "9876")

    # --- All done ---
    f.close()

if __name__ == '__main__':

    write_example('/tmp/python/DataExchange-example5.h5')
#=====================================================================
#
#=====================================================================
```

Listing 7: Python Code for Data Exchange full implementation for tomography - includes provenance

```
#=====================================================================
# Sample Python code to write Data Exchange Format
#
# Date: 2013-04-21
#
#=====================================================================

import h5py
import numpy as np

def write_example(filename):

    # --- prepare data ---

    # Generate fake raw data
    rawdata = np.ones(180 * 256 * 256, np.uint16).reshape(180, 256, 256)
    rawdata_white = np.ones(2 * 256 * 256, np.uint16).reshape(2, 256, 256)
    rawdata_dark = np.zeros(10 * 256 * 256, np.uint16).reshape(10, 256, 256)

    # Generate fake normalized data
    normalizeddata = np.ones(180 * 256 * 256, \
                             np.float64).reshape(180, 256, 256)

    # Generate fake reconstructed data
    reconstructeddata = np.ones(256 * 256 * 256, \
                                np.float64).reshape(256, 256, 256)

    # Generate fake provenance data
#    provenancedata = np.chararray((10, 10, 10))

    # x, y and z ranges
    x = np.arange(256)
    y = np.arange(256)
    z = np.arange(180);

    # Fabricated theta values
    theta = (z / float(180)) * 180.0
    theta_white = (0.0, 180.0)
    theta_dark = (0.0, 0.0, 0.0, 0.0, 0.0, 180.0, 180.0, 180.0, 180.0, 180.0)

    # --- create file ---
```

```
# Open HDF5 file
f = h5py.File(filename, 'w')

# Create basic definitions in root
ds = f.create_dataset('implements', \
                data = "exchange:exchange_2:exchange_3:measurement")

# --- exchange definition ---

# Exchange HDF5 group
# /exchange
exchangeGrp = f.create_group("exchange")

# Create core HDF5 dataset in exchange group for 180 deep stack
# of x,y images /exchange/data
ds = exchangeGrp.create_dataset('data', data = rawdata, \
                                compression='gzip', compression_opts=4)

ds.attrs['description'] = "transmission"
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta:y:x"
ds1 = exchangeGrp.create_dataset('title', \
                                 data = "tomography raw projections")

# Create HDF5 dataset in exchange group for dark data
# /exchange/data_dark
ds = exchangeGrp.create_dataset('data_dark', \
                                data = rawdata_dark, \
                                compression='gzip', compression_opts=4)
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta_dark:y:x"

# Create HDF5 dataset in exchange group for white data
# /exchange/data_white
ds = exchangeGrp.create_dataset('data_white', \
                                data = rawdata_white, \
                                compression='gzip', compression_opts=4)
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta_white:y:x"

# Create HDF5 dataset in exchange group for theta
# /exchange/theta
```

```
ds = exchangeGrp.create_dataset('theta', data = theta)
ds.attrs['units'] = "degrees"

# Create HDF5 dataset in exchange group for theta_dark
# /exchange/theta_dark
ds = exchangeGrp.create_dataset('theta_dark', data = theta_dark)
ds.attrs['units'] = "degrees"

# Create HDF5 dataset in exchange group for theta_white
# /exchange/theta_white
ds = exchangeGrp.create_dataset('theta_white', data = theta_white)
ds.attrs['units'] = "degrees"

# Exchange HDF5 group
# /exchange_2
# this will be the out_put of the normalization process
exchange2Grp = f.create_group("exchange_2")
ds2 = exchange2Grp.create_dataset('title', \
                        data = "tomography normalized projections")

# Create core HDF5 dataset in exchange group for 180 deep stack
# of x,y images /exchange_2/data
ds = exchange2Grp.create_dataset('data', data = normalizeddata, \
                                 compression='gzip', \
                                 compression_opts=4)
ds.attrs['units'] = "counts"
ds.attrs['axes'] = "theta:y:x"
# Create HDF5 dataset in exchange_2 group for theta
# /exchange_2/theta
ds = exchange2Grp.create_dataset('theta', data = theta)
ds.attrs['units'] = "degrees"

# Exchange HDF5 group
# /exchange_3
# this will be the out_put of the reconstruction process
exchange3Grp = f.create_group("exchange_3")
ds3 = exchange3Grp.create_dataset('title', \
                                  data = "tomography reconstructions")

# Create core HDF5 dataset in exchange group for 180 deep stack of x,y ima
# /exchange_3/data
ds = exchange3Grp.create_dataset('data', \
                                 data = reconstructeddata, \
```

```
                                    compression='gzip', \
                                    compression_opts=4)
ds.attrs['units'] = "density"
ds.attrs['axes'] = "z:y:x"

# Create HDF5 group measurement
# /measuremen
measurementGrp = f.create_group("measurement")

# Create HDF5 subgroup
# /measurement/instrument
instrumentGrp = measurementGrp.create_group("instrument")

# Create HDF5 subgroup
# /measurement/instrument/source
sourceGrp = instrumentGrp.create_group("source")
sods1 = sourceGrp.create_dataset('name', data = "APS")
sods2 = sourceGrp.create_dataset('date_time', data = "2012-07-31T21:15:22+

sods3 = sourceGrp.create_dataset('beamline', data = "2-BM")
sods4 = sourceGrp.create_dataset('current', data = 401.199, dtype='d')
sods4.attrs['units'] = "mA"
sods5 = sourceGrp.create_dataset('energy', data = 7.0, dtype='d')
sods5.attrs['units'] = "GeV"
sods6 = sourceGrp.create_dataset('mode', data = "TOPUP")

# Create HDF5 subgroup
# /measurement/instrument/attenuator
attenuatorGrp = instrumentGrp.create_group("attenuator")
ats1 = attenuatorGrp.create_dataset('thickness', data = 1e-3, \
                                    dtype='d')
ats2 = attenuatorGrp.create_dataset('type', data = "Al")

# Create HDF5 subgroup
# /measurement/instrument/monochromator
monochromatorGrp = instrumentGrp.create_group("monochromator")
mds1 = monochromatorGrp.create_dataset('type', data = "Multilayer")
mds2 = monochromatorGrp.create_dataset('energy', data = 19.26, \
                                       dtype='d')
mds2.attrs['units'] = "keV"
mds3 = monochromatorGrp.create_dataset('energy_error', data = 1e-3, \
                                       dtype='d')
mds3.attrs['units'] = "keV"
```

```
mds4 = monochromatorGrp.create_dataset('mono_stripe', data = "Ru/C")

# Create HDF5 subgroup
# /measurement/instrument/detector
detectorGrp = instrumentGrp.create_group("detector")

# define the detector subgroup members of instrument
dds1 = detectorGrp.create_dataset('manufacturer', \
                                  data = "CooKe Corporation")
dds2 = detectorGrp.create_dataset('model', data = "pco dimax")
dds3 = detectorGrp.create_dataset('serial_number', data = "1234XW2")
dds4 = detectorGrp.create_dataset('bit_depth', data = 12, dtype='i')

# for x_pixel_size and y_pixel_size if the attributes units is not
# specified then these are in meters
dds5 = detectorGrp.create_dataset('x_pixel_size', data = 6.7e-6, \
                                  dtype='f')
dds6 = detectorGrp.create_dataset('y_pixel_size', data = 6.7e-6, \
                                  dtype='f')
dds7 = detectorGrp.create_dataset('x_dimentions', data = 2048, \
                                  dtype='i')
dds8 = detectorGrp.create_dataset('y_dimentions', data = 2048, \
                                  dtype='i')
dds9 = detectorGrp.create_dataset('x_binning', data = 1, dtype='i')
dds10 = detectorGrp.create_dataset('y_binning', data = 1, dtype='i')

# for operating_temperature if the attributes units is not
# specified then this is in K
dds11 = detectorGrp.create_dataset('operating_temperature', \
                                   data = 270, dtype='f')

# for exposure_time the attributes units is specified as ms
dds12 = detectorGrp.create_dataset('exposure_time', data = 170, \
                                   dtype='d')
dds12.attrs['units'] = "ms"

dds13 = detectorGrp.create_dataset('frame_rate', data = 3, dtype='i')
dds14 = detectorGrp.create_dataset('output_data', data = "/exchange")

roiGrp = detectorGrp.create_group("roi")
rds1 = roiGrp.create_dataset('name', data = "center third")
rds2 = roiGrp.create_dataset('x1', data = 256, dtype='i')
rds3 = roiGrp.create_dataset('y1', data = 256, dtype='i')
```

```
rds4 = roiGrp.create_dataset('x2', data = 1792, dtype='i')
rds5 = roiGrp.create_dataset('y2', data = 1792, dtype='i')

objectiveGrp = detectorGrp.create_group("objective")
ods1 = objectiveGrp.create_dataset('manufacturer', data = "Zeiss")
ods2 = objectiveGrp.create_dataset('model', \
                                   data = "Plan-NEOFLUAR 1004-072")
ods3 = objectiveGrp.create_dataset('magnification', \
                                   data = 20, dtype='d')
ods4 = objectiveGrp.create_dataset('numerical_aperture', \
                                   data = 0.5, dtype='d')


scintillatorGrp = detectorGrp.create_group("scintillator")
sds1 = scintillatorGrp.create_dataset('manufacturer', data = "Crytur")
sds2 = scintillatorGrp.create_dataset('serial_number', data = "12")
sds3 = scintillatorGrp.create_dataset('name', data = "YAG polished")
sds4 = scintillatorGrp.create_dataset('type', data = "YAG on YAG")
sds5 = scintillatorGrp.create_dataset('scintillating_thickness', \
                                      data = 5e-6, dtype='d')
sds6 = scintillatorGrp.create_dataset('substrate_thickness', \
                                      data = 1e-4, dtype='d')


# Create HDF5 subgroup
# /measurement/sample
sampleGrp = measurementGrp.create_group("sample")
sads1 = sampleGrp.create_dataset('name', data = "Hornby_b")
sads2 = sampleGrp.create_dataset('description', data = "test sample")
sads3 = sampleGrp.create_dataset('preparation_date', \
                                 data = "2012-07-31T21:15:22+0600")
sads4 = sampleGrp.create_dataset('chemical_formula', data = "unknown")
sads5 = sampleGrp.create_dataset('mass', data = 0.25, dtype='d')
sads5.attrs['units'] = "g"
sads7 = sampleGrp.create_dataset('enviroment', data = "air")
sads8 = sampleGrp.create_dataset('temperature', data = 120.0, \
                                 dtype='d')
sads8.attrs['units'] = "Celsius"
sads9 = sampleGrp.create_dataset('temperature_set', data = 130.0, \
                                 dtype='d')
sads9.attrs['units'] = "Celsius"

# Create HDF5 subgroup
```

```
# /measurement/sample/geometry
geometryGrp = sampleGrp.create_group("geometry")

# Create HDF5 subgroup
# /measurement/sample/geometry/translation
translationGrp = geometryGrp.create_group("translation")
trds1 = translationGrp.create_dataset('distances', \
                                      data = [0, 0, 0], dtype='d')
trds1.attrs['axes'] = "z:y:x"
trds1.attrs['units'] = "m"

# Create HDF5 subgroup
# /measurement/sample/experimenter
experimenterGrp = sampleGrp.create_group("experimenter")
exrds1 = experimenterGrp.create_dataset('name', data = "John Doe")
exrds2 = experimenterGrp.create_dataset('role', data = "Project PI")
exrds3 = experimenterGrp.create_dataset('affiliation', \
                    data = "University of California, Berkeley")
exrds4 = experimenterGrp.create_dataset('address', \
                    data = "EPS UC Berkeley CA 94720 4767 USA")
exrds5 = experimenterGrp.create_dataset('phone', \
                    data = "+1 123 456 0000")
exrds6 = experimenterGrp.create_dataset('e-mail', \
                    data = "johndoe@berkeley.edu")
exrds7 = experimenterGrp.create_dataset('facility_user_id', \
                    data = "a123456")

# Create HDF5 subgroup
# /measurement/sample/experiment
experimentGrp = sampleGrp.create_group("experiment")
exds1 = experimentGrp.create_dataset('proposal', data = "1234")
exds2 = experimentGrp.create_dataset('activity', data = "e11218")
exds3 = experimentGrp.create_dataset('safety', data = "9876")

# Create HDF5 group provenance
# /provenance
provenanceGrp = f.create_group("provenance")

# Create core provenance compound dataset in /provenance
# dt=h5py.special_dtype(vlen=str)
my_dtype = np.dtype([('actor', np.str_, 64), ('start_time', np.str_, 64),
ds = provenanceGrp.create_dataset('process', (10,1), dtype=my_dtype)
ds [0] = ('gridftp', '2012-07-31T21:15:22+0600', '2012-07-31T21:15:23+0600
```

```
ds [1] = ('gridftp', '2012-07-31T21:15:23+0600', '2012-07-31T21:15:24+0600
ds [2] = ('gridftp', '2012-07-31T21:15:25+0600', '2012-07-31T22:15:22+0600
ds [3] = ('norm', '2012-07-31T22:15:23+0600', '2012-07-31T22:30:22+0600',
ds [4] = ('rec', '2012-07-31T22:30:23+0600', '2012-07-31T22:50:22+0600', '
ds [5] = ('convert', '2012-07-31T22:50:23+0600', '', 'RUNNING', 'convert m
ds [6] = ('gridftp', '', '', 'QUEUED', 'convert message','/provenance/grid

# Create HDF5 subgroup
# /provenance for process_1
gridftpGrp = provenanceGrp.create_group("gridftp")
grds1 = gridftpGrp.create_dataset('name', data = "gridftp")
grds2 = gridftpGrp.create_dataset('version', data = "1.0")
grds3 = gridftpGrp.create_dataset('source_URL', \
                                  data = "gsiftp://host1/path")
grds4 = gridftpGrp.create_dataset('dest_URL', \
                                  data = "gsiftp://host2/path")

# Create HDF5 subgroup
# /provenance for process_2
normalizeGrp = provenanceGrp.create_group("norm")
nods1 = normalizeGrp.create_dataset('name', data = "normalize")
nods2 = normalizeGrp.create_dataset('version', data = "1.0")
nods3 = normalizeGrp.create_dataset('input_data', data = "/exchange")
nods4 = normalizeGrp.create_dataset('output_data', data = "/exchange_2")

# Create HDF5 subgroup
# /provenance for process_3
reconstructGrp = provenanceGrp.create_group("rec")
reds1 = reconstructGrp.create_dataset('name', data = "reconstruct")
reds2 = reconstructGrp.create_dataset('version', data = "1.0")
reds3 = reconstructGrp.create_dataset('input_data', data = "/exchange_2")
reds4 = reconstructGrp.create_dataset('output_data', data = "/exchange_3")
reds5 = reconstructGrp.create_dataset('reconstruction_slice_start', \
                                      data = 0, dtype='i')
reds6 = reconstructGrp.create_dataset('reconstruction_slice_end', \
                                      data = 147, dtype='i')
reds7 = reconstructGrp.create_dataset('rotation_center', \
                                      data = 1048.50, dtype='d')

algorithmGrp = reconstructGrp.create_group("algorithm")
alds1 = algorithmGrp.create_dataset('name', data = "GRIDREC")
alds2 = algorithmGrp.create_dataset('version', data = "1.0")
alds3 = algorithmGrp.create_dataset('filter', data = "Parzen")
```

```
    alds4 = algorithmGrp.create_dataset('padding', data = 0.5, dtype='d')

    # Create HDF5 subgroup
    # /provenance for process_4
    exportGrp = provenanceGrp.create_group("export")
    exds1 = exportGrp.create_dataset('name', data = "convert2tiff")
    exds2 = exportGrp.create_dataset('version', data = "1.0")
    exds3 = exportGrp.create_dataset('input_data', data = "/exchange_3")
    exds4 = exportGrp.create_dataset('output_data', data = "file://host3/path"
    exds5 = exportGrp.create_dataset('output_data_format', data = "TIFF")
    exds6 = exportGrp.create_dataset('output data scaling max', \
                                     data = 0.005, dtype='d')
    exds7 = exportGrp.create_dataset('output data scaling min', \
                                     data = -0.00088, dtype='d')

    # Create HDF5 subgroup
    # /provenance for process_5
    gridftpGrp = provenanceGrp.create_group("gridftp_2")
    grds1 = gridftpGrp.create_dataset('name', data = "gridftp")
    grds2 = gridftpGrp.create_dataset('version', data = "1.0")
    grds3 = gridftpGrp.create_dataset('source_URL', \
                                      data = "gsiftp://host3/path")
    grds4 = gridftpGrp.create_dataset('dest_URL', \
                                      data = "gsiftp://host4/path")

    # --- All done ---
    f.close()

if __name__ == '__main__':

    write_example('/tmp/python/DataExchange-example6.h5')
#=====================================================================
#
#=====================================================================
```

8.2 Creating Data Exchange files using C++

A Appendix

A.1 Default units for Data Exchange entries

The default units for Data Exchange entries follow the CXI entries definition, i.e. are SI based units (see table **??**) unless the ”units” attribute is specified. Data Exchange prefers to use the default SI based units whenever possible.

Table 34: 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	Ω
absorbed dose	gray	Gy
area	square meter	m^2
volume	cubic meter	m^3

A.1.1 Exceptions

Angles are always defined in degrees *not* in radians and use the abbreviation ”degree”.

A.1.2 Times and Dates

Times and 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.

A.2 Geometry

A.2.1 Coordinate System

The Data Exchange uses the same CXI coordinate system. This is a right handed system with the z axis parallel to the X-ray beam, with the positive z direction pointing away from the light source, in the downstream direction. The y axis is vertical with the positive direction pointing up, while the x axis is horizontal completing the right handed system (see Fig. **??**). The origin of the coordinate system is defined by the point where the X-ray beam meets the sample.

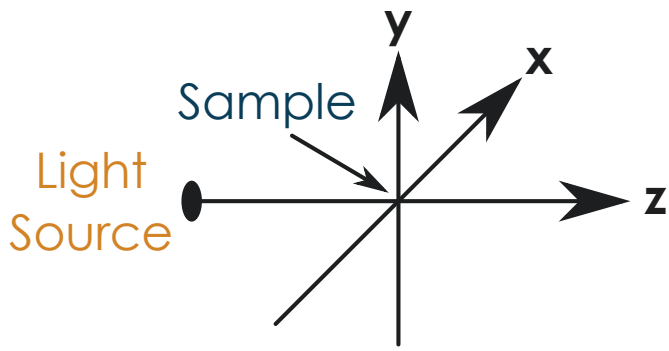

Figure 9: The coordinate system used by CXI. The intersection of the X-ray beam with the sample define the origin of the system. The z axis is parallel to the beam and points downstream.

A.2.2 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.

List of Figures